Calcul d'itinéraire multimodal et multiobjectif en milieu urbain

Tristram Gräbener

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Université Toulouse 1 Capitole (UT1 Capitole)

Discipline ou spécialité :

Informatique

Présentée et soutenue par :
Tristram Gräbener

le : lundi 15 novembre 2010

Titre :

Calcul d'itinéraire multimodal et multiobjectif en milieu urbain

---

Ecole doctorale :
Mathématiques Informatique Télécommunications (MITT)

Unité de recherche :
Institut de Recherche en Informatique de Toulouse (IRIT)

Directeur(s) de Thèse :
Yves Duthen, Professeur, Université Toulouse 1 Capitole
Alain Berro, Maître de Conférence, Université Toulouse 1 Capitole

Rapporteurs :
Xavier Gandibleux, Professeur, Université de Nantes
Jin-Kao Hao, Professeur, Université d'Angers

Autre(s) membre(s) du jury

Christian Artigues, Chargé de recherche 1ère classe, LAAS-CNRS
Cyrille Granié, Ingénieur, SODIT

Remerciements

Vouloir remercier les personnes qui ont participé, influencé ou juste soutenu ma thèse est un exercice difficile. Cela suppose de faire le tri entre toutes les personnes qu'on a cotoyées en espérant n'avoir oublié personne. Pour contourner ce problème, je vais remercier des groupes d'amis, sans donner de nom en particulier.

Je commence par une exception en remerciant Yves pour m'avoir accueilli et d'avoir été mon directeur de thèse. La deuxième (et avant-avant-dernière) exception est Alain pour son encadrement, pour avoir supporté mes idées buttées quand il s'agit de programmation et mon manque d'intérêt dans le championnat de rugby.

Merci aux rapporteurs, Xavier Gandibleux et Jin-Kao Hao d'avoir montré un intérêt pour mon sujet de thèse.

Évidemment merci à ma famille qui m'a toujours encouragé à faire une thèse et leur volonté à essayer de comprendre plus concrètement ce que je fais.

Merci à toute la ME-310 pour l'ambiance détendue, les débats stériles et les expéditions vers Saint-Pierre ou le Txus. Il ne faut pas non plus oublier les permanents pour faciliter la bonne ambiance et pour leurs conseils. À cette vie quotidienne ont également contribué les filles de la cafétéria avec plats chauds et assiettes anglaises froides.

Il y a également les thésards d'autres disciplines rencontrés lors des formations ou lors des sorties AMESAT. Je pense en particulier aux trolls norvégiens.

Parmi mes amis les plus anciens il y a les nantais et les *gasy* qui ont pu suivre de plus ou moins loin ces trois années et que j'ai toujours eu plaisir à revoir régulièrement.

Parmi mes amis les plus récents, il y a les capoeiristes de Ginga Nagô qui m'ont permis de faire de beaux voyages, avoir des courbatures presque toutes les semaines et rajouter la fatigue physique à la fatigue intellectuelle.

Enfin, merci à la Sodit et la région Midi-Pyrénées pour avoir montré un intérêt dans les problématiques de cette thèse et pour m'avoir financé pendant trois ans.

Et si après ça, j'ai encore oublié quelqu'un, merci à tous !

PS : Je remercie également tous ceux qui ont lu le manuscrit après l'envoi aux rapporteurs et qui ont trouvé une quantité impressionante de fautes.

Résumé

Motivations

Avec une prise de conscience environnementale croissante et avec l'augmentation des coûts de l'énergie, de plus en plus de personnes tendent à utiliser les transports en commun ou des modes de transports doux tels que la marche ou le vélo.

Cependant, un seul mode de transport ne peut pas couvrir tous les besoins. De ce fait, la combinaison de différents modes de transport peut se révéler être une solution particulièrement intéressante. Par exemple, l'usager peut conduire une voiture jusqu'à une gare ou encore marcher entre deux stations pour éviter un changement supplémentaire.

Trouver le *meilleur* chemin multimodal pour une personne donnée est une tâche difficile. Chaque personne aura des préférences différentes concernant la durée du trajet, le coût monétaire, la pollution engendrée, le nombre de changements, etc. De plus, les préférences d'un même usager dépendront des circonstances. Par exemple s'il pleut, il ne va pas prendre le vélo et s'il a des bagages encombrants, il préférera probablement éviter des changements. L'optimisation *multiobjectif* permet de proposer à l'utilisateur plusieurs solutions dites *équivalentes*. Ainsi l'utilisateur choisira l'itinéraire qui lui convient en fonction de ses préférences à un moment donné.

Problématiques

Le problème principal à résoudre est donc le problème du plus court chemin multiobjectif de point à point dépendant du temps. L'enjeu est d'être capable d'avoir des résultats de l'ordre de la seconde à l'échelle d'une grande agglomération afin de pouvoir envisager une application réelle. Nous considérons en effet que c'est un temps tout à fait acceptable pour un site web proposant le calcul ou encore pour un outil de navigation individuel.

Toujours dans cette optique d'application réelle, une attention particulière a été portée sur la simplicité et la généricité des approches proposées. Nous ne nous restreignons pas à un nombre prédéfini de modes de transport ou d'objectifs.

Étant donné que le front de Pareto peut être exponentiel, nous nous sommes également posé la question de comment sélectionner un nombre plus restreint de solutions pour ne pas noyer l'utilisateur dans un grand nombre de solutions trop similaires.

Approches proposées

Après un état de l'art sur les plus courts chemins dépendants du temps, des plus courts chemins multiobjectifs et le transport multimodal, nous proposons un modèle qui permet d'exploiter des algorithmes traditionnels de parcours de graphes tel que l'algorithme de Dijkstra avec très peu d'adaptations. Nous avons de plus fait attention à garder une forte généricité afin de pouvoir prendre en compte de nouveaux modes de transport tels que les vélos en libre service et un grand nombre d'objectifs généralement négligés tels que les émissions de CO_2 ou encore le dénivelé à vélo.

La généricité du modèle nous permet d'appliquer directement l'algorithme de Martins en ne l'adaptant que légèrement pour gérer la dépendance au temps pour calculer le plus court chemin multiobjectif. D'autres algorithmes réputés pour leurs performances théoriques ou expérimentales ont été adaptés au cas multiobjectif ou à la dépendance du temps. Nous avons également proposé des heuristiques simples permettant de garder le temps de calcul de l'ordre de la seconde.

Enfin, comme dans certains cas il peut y avoir une centaine de solutions possibles, nous avons envisagé plusieurs approches pour limiter le nombre de solutions retournées. Ces approches peuvent s'appliquer lors de la modélisation, mais aussi lors de la résolution ou encore *a posteriori*.

Applications

Les algorithmes ont été appliqués avec succès sur les villes de San Francisco, Los Angeles et Rennes. Les performances sont suffisantes pour envisager une application réelle.

Les modes de transport considérés sont la voiture, le vélo, la marche, les transports en commun et les vélos en libre service. Les objectifs que nous avons considérés sont le temps de trajet, le nombre de changements de modes, le nombre de changements de ligne au sein d'un même mode de transport, le dénivelé positif à vélo et les émissions de CO_2.

Table des matières

Notations

$\mathcal{N}$	Ensemble des nœuds
n	Nombre de nœuds dans le graphe
$\mathcal{A}$	Ensemble des arcs
m	Nombre d'arcs dans le graphe
$\mathcal{G}$	Graphe
u	Exemple de nœud
v	Exemple de nœud
c	Poids (ou coût) d'un arc ou d'un chemin
$\mathcal{C}$	Ensemble des poids (ou coûts) associés aux arcs d'un graphe
Γ^+	Ensemble des successeurs d'un nœud
Γ^-	Ensemble des prédécesseurs d'un nœud
$\mathcal{F}^t$	Temps de parcours d'un arc
$\mathcal{F}^c$	Fonction de coût d'un arc
q	Nombre d'intervalles de temps
Ω	Espace de décision
k	Nombre d'objectifs
$\mathcal{P}^*$	Ensemble Pareto-optimal dans l'espace de décision
$\mathcal{PF}^*$	Front de Pareto, dans l'espace des solutions

Figures

Algorithmes

Introduction

People of Earth, your attention, please [...] As you will no doubt be aware, the plans for development of the outlying regions of the Galaxy require the building of a hyperspatial express route through your star system. And regrettably, your planet is one of those scheduled for demolition. The process will take slightly less than two of your Earth minutes. Thank you.

— Douglas Adams, The Hitchhiker's Guide to the Galaxy

Situation actuelle des transports en France

Un bilan écologique catastrophique pour la voiture individuelle

Premier émetteur de CO_2 en France

Le transport est responsable en France de 34,3% des émissions de CO_2 en 2007, loin devant le résidentiel et le tertiaire (21,1%), de l'industrie (19,4%) ou de la production d'électricité et de chaleur (12%)[1].

Entre 1990 et 2007 les émissions de CO_2 liées au transport ont augmenté de 27%, réduisant ainsi à néant les efforts de réduction faits par l'habitat et l'industrie.

Même si le transport aérien émet plus de CO_2 par kilomètre et par passager par rapport à une voiture avec deux passagers, il n'est heureusement pas utilisé régulièrement par tout le monde. De ce fait il ne représente donc que 4,7 millions de tonnes de CO_2 sur les 142,8 millions émis par les transports en France.

Le transport de marchandises ne représente que 18,1 millions de tonnes de CO_2 sur les 129,8 millions émis par le transport routier.

La voiture *individuelle* est donc de très loin (78%) le premier responsable des émissions de CO_2 liées au transport.

[1] Service de l'Observation et des Statistiques (SOeS) du Commissariat Général au Développement Durable (CGDD)

À l'échelle de toutes les émissions de CO_2 de la France, la voiture *individuelle* représente donc toujours la première cause avec 23,7% des émissions en 2006.

Autres sources de problèmes

La voiture est également responsable d'un grand nombre d'autres problèmes. On estime que 80% du bruit d'une ville provient de la voiture[2]. Paris est recouverte à 25% par les voiries[3]. En 2009, les accidents de la route ont tué 4 252 personnes et en ont blessé 83 111[4].

Le coût de possession annuel d'une voiture est estimé en 2004 à 4 273 euros selon l'INSEE[5], tandis que les routes coûtent aux collectivités 26 Millards d'euros[6].

Des alternatives efficaces

Aucune amélioration significative de l'automobile

Qu'il s'agisse de voitures hybrides, à hydrogène, à agrocarburants ou encore totalement électriques, les progrès technologiques promis par l'industrie automobile sont dérisoires, et présenteront de nouveaux problèmes (énergie nécessaire à la production de l'électricité ou de l'hydrogène, concurrence des agrocarburants avec l'agriculture alimentaire, manque de surface et appauvrissement des sols, rareté de composants tels que le lithium).

À titre de comparaison, une voiture électrique consomme environ 20kWh pour 100km. À raison de 100g de CO_2 par kWh (moyenne française ; la moyenne européenne est de 400g/kWh), une telle voiture émet donc 20g de CO_2 par km en France et 80g en moyenne en Europe ![1].

Des transports existants efficaces

Un autobus urbain de 100 places émet environ 800g/km. En optimisant le remplissage il est donc possible d'obtenir des meilleurs résultats avec des technologies largement utilisées qu'avec une technologie qui est encore à un stade embryonnaire. Le métro et le tramway émettent en France respectivement 4g et 3g de CO_2 par passager et par kilomètre[2].

---

[2] Ademe : Agence de l'Environnement et de la Maîtrise de l'Energie

[3] `http://www.paris.fr/`

[4] `http://www.preventionroutiere.asso.fr/accidentologie_securite_routiere.aspx`

[5] `http://www.insee.fr/fr/ffc/docs_ffc/ip1039.pdf`

[6] `http://www.rprudhomme.com/resources/2009+co$C3$BBts+de+la+route.pdf`

De plus la recherche sur de nouveaux matériaux, la récupération d'énergie lors du freinage et l'optimisation du remplissage des transports en commun devraient d'autant plus améliorer ces excellents chiffres.

Des transports presque parfaits

Bien évidemment bien plus écologiques sont les modes doux tels que la marche ou le vélo qui n'émettent absolument aucune émission. En effet le carbone émis lors de la respiration est issu de notre alimentation qui est — ou du moins devrait — être renouvelable. Si nous prenions en plus en compte les émissions liées à la fabrication de véhicules, alors les transports doux seraient d'autant plus vertueux.

Un problème d'information

Une méconnaissance des réseaux

La situation est paradoxale. La plupart des personnes intérrogées répondent qu'elles abandonneraient la voiture si les transports en commun étaient mieux développées. Pourtant une part considérable de la population en France est desservie par les transports en communs et se situe à moins de deux kilomètres d'un arrêt de transports en communs (soit moins de dix minutes à vélo). Il s'agit donc bien souvent d'une méconnaissance des réseaux (« je ne savais pas qu'il y avait telle ligne »), d'un manque de volonté de mieux les connaître (« je n'ai pas envie d'apprendre par cœur l'horaire du bus »), de préjugés (« le bus c'est lent »[7]).

Ainsi, très peu de personnes savent qu'il est possible de prendre un bus RATP au prix d'un simple ticket (1€70 en juillet 2010) pour aller de l'aéroport d'Orly à Paris[8].

Cette situation est encore plus criante dans le communes rurales. Ainsi le réseau des 62 lignes régulières d'autocar de la Haute-Garonne est inconnu pour la plupart des Toulousains alors qu'il permet d'accéder à pratiquement toutes les communes du département[9].

---

[7] Pourtant de nombreux réseaux de bus sudaméricains tels que celui de Curitiba, le *Transmilenio* de Bogota, le trolleybus de Quito ou encore la *Metrovia* de Guayaquil feraient pâlir toutes les lignes de tramway en France et même certaines lignes de métro en termes de passagers transportés et de vitesse commerciale

[8] Lignes 183 et 285

[9] `http://www.haute-garonne.fr/upload/pdf_arcenciel/carte_lignes_arc_en_ciel.pdf`

Trop de possibilités

Finalement, l'offre des transports en commun est pénalisée par sa richesse. Ainsi la RATP et la SNCF proposent cinq cartes différentes pour l'Île-de-France : réseau de bus, réseau de métro, réseau de RER et réseau de Transilien (trains de banlieue) et le réseau noctilien (bus de nuit). À celà il faut rajouter les réseaux de bus départementaux.

De ce fait, l'utilisateur a tendance à se rabattre sur un seul mode de transport et n'envisagera pas les autres devant la complexité de se représenter tout le réseau multimodal. Il est donc parfaitement compréhensible que les trajets empruntés ne combinent pas plusieurs modes de transport.

Objectifs de cette thèse

Malgré cette richesse de l'offre de transport en commun et l'impact écologique de la voiture, celle-ci a représenté 81,9% des kilomètre-passager effectués[1] en 2009. Il existe donc une marge de manœuvre colossale pour réduire la part de la voiture dans nos déplacements et donc de significativement réduire les émissions de gaz à effet de serre et les autres nuisances.

Pour cela il est bien évidemment nécessaire de développer les offres de transport en commun ainsi que les réseaux cyclables. Il serait cependant irréaliste de considérer qu'il existe un mode de transport qui serait capable de totalement remplacer la voiture tellement les situations varient entre deux personnes.

La combinaison de plusieurs modes de transport permettrait à une fraction des automobilistes de se reporter vers des modes de transport plus économiques et plus écologiques. Ainsi en utilisant le vélo il sera possible d'atteindre une gare située à 2 km de sa maison et donc compenser l'absence de transports en commun denses.

Cependant une part des automobilistes continue à utiliser la voiture par méconnaissance des possibilités d'itinéraires. En effet l'absence d'un arrêt de bus à proximité décourage l'utilisation des transports en commun et la trop grande distance décourage de l'utilisation du vélo. Pourtant la combinaison du vélo et des transports en commun peut se révéler être un excellent compromis. Les possibilités d'itinéraires deviennent d'autant plus nombreuses que se développent de nouveaux modes de transport tels que les vélos en libre service. Il est donc indispensable de développer un outil capable de présenter plusieurs propositions à un utilisateur pour qu'il trouve un compromis qui lui corresponde.

Nous proposons au cours de cette thèse un système ayant pour ambition de présenter à un utilisateur plusieurs itinéraires entre deux points afin qu'il y trouve un compromis intéressant correspondant à sa propre situation et aux efforts qu'il est prêt à consentir.

Nous pensons en effet que tout utilisateur à qui l'on présente de nouveaux trajets permettant d'atteindre la destination envisagera sérieusement d'abandonner, au moins en partie sur quelques trajets, la voiture. Ainsi, même en changeant le comportement d'une partie des automobilites, il sera possible de réduire les émissions de CO_2 sans effectuer le moindre investissement.

Ainsi, si l'impact sur l'organisation des transports reste très modeste, nous espérons que l'adoption d'un tel système permettra de contribuer au recul de la voiture.

Scientifiquement, cette thèse est plus ambitieuse, puisque nous voulons proposer une approche globale au problème du plus court chemin multimodal et multiobjectif qui puisse être appliquée sur des instances réelles de grande taille avec des performances raisonnables. Nous présentons les différents critères que nous souhaitons prendre en compte dans le **tableau 2.1** et montrons qu'aucune étude antérieure n'a, à notre connaissance, proposé une approche pour satisfaire tous ces critères.

Organisation du manuscrit

Nous commençons par présenter les travaux existants dans le domaine du calcul d'itinéraire dans un cadre dépendant du temps mais également lorsque nous désirons optimiser plusieurs critères simultanément.

Le deuxième chapitre étudie théoriquement et expérimentalement la nature des objectifs à optimiser afin d'avoir une meilleure compréhension des limites d'application.

Le troisième chapitre présente le modèle que nous proposons. Celui-ci est particulièrement simple à appréhender sans pour autant limiter les possibilités de combinaison de modes de transport ou de critères à optimiser.

Les chapitres 4 et 5 présentent le calcul d'itinéraire à proprement parler. Dans un premier temps il s'agit uniquement de trouver le trajet le plus rapide et dans un deuxième temps en optimisant d'autres critères.

L'approche multiobjectif que nous avons adoptée propose plusieurs alternatives à l'utilisateur. Cependant dans certains cas trop de solutions peuvent nuire à l'intérêt du système. Nous nous intéressons donc dans le sixième chapitre à la sélection d'un nombre réduit de solutions de qualité.

Enfin, le septième et dernier chapitre propose quelques variantes aux problèmes que nous avons traités.

Première partie :

État de l’art et problématique

Chapitre 1
État de l'art

Dans le cochon, tout est bon.

— Adage populaire français

Introduction

Ce chapitre a pour but de rappeler les bases requises sur les graphes pour situer nos travaux.

Deux axes théoriques sont fondamentaux pour s'intéresser à notre sujet : le plus court chemin dynamique et le plus court chemin multiobjectif. Cependant avant de les présenter, nous décrivons leurs bases communes. Dans un premier temps nous présentons les généralités autour des graphes. Dans un deuxième temps le plus court chemin dans un graphe, en nous focalisant sur les deux algorithmes les plus utilisés — à savoir celui de Dijkstra et celui de Bellman. En partant de ces algorithmes, nous verrons comment traiter le cas lorsque le graphe varie en fonction du temps. Enfin, nous nous intéresserons au plus court chemin multiobjectif.

Ces bases théoriques ayant été posées, nous étudierons les rares approches existantes qui traitent du plus court chemin multimodal afin de mieux situer nos axes de recherche.

1.1 Généralités sur les graphes

Dans un article publié en 1736, Leonhard Euler démontre qu'il est impossible d'emprunter une et une seule fois les sept ponts de Königsberg (aujourd'hui Kaliningrad). Pour cela il introduit une structure de données qui sera appelée plus tard *graphe*.

Un graphe est une structure de données très simple utilisée dans de nombreux domaines tels que les télécommunications, la planification, l'électronique, les transports ou encore la théorie de la complexité.

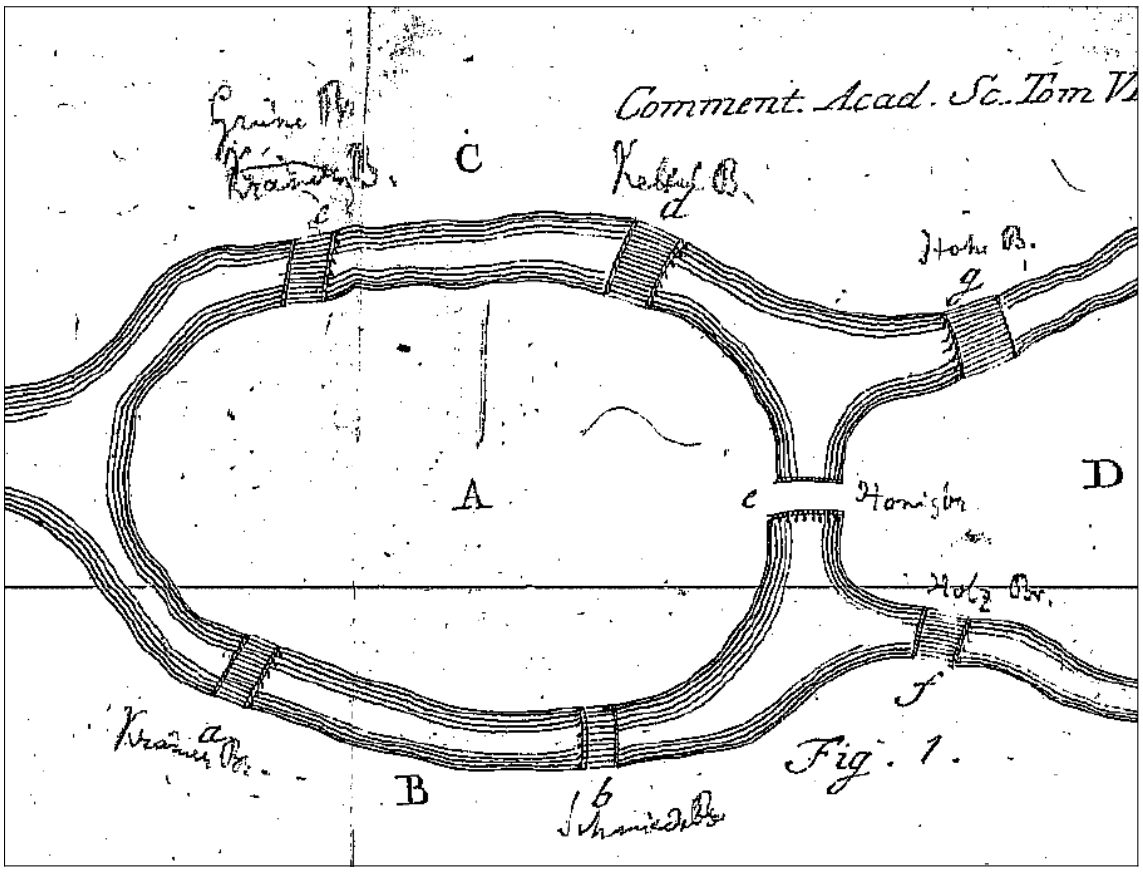


Figure 1.1 Les sept ponts de Königsberg. Euler démontra qu'il est impossible de traverser une et une seule fois tous les ponts et introduisit la notion de *graphe*

Google Scholar référence près de cinq millions d'articles comportant le mot *graph* dans le titre. Face à l'étendue du domaine il ne serait donc pas possible de présenter l'ensemble de la théorie des graphes. Nous nous focaliserons donc dans cet état de l'art sur le problème particulier du plus court chemin. Après avoir défini les notations, nous présenterons les principaux algorithmes en nous intéressant en particulier à leurs conditions d'applications.

1.1.1 Notations

En simplifiant à l'extrême, un graphe définit l'existence d'une relation entre objets tels qu'une ligne entre deux stations de métro, une relation d'amitié dans un réseau social ou encore une rue entre deux carrefours.

Nœuds et arcs

Les objets sont appelés *nœuds* (*nodes*) ou *sommets* (*vertices*) et les relations *arcs* (*edges*) ou *arêtes* (*links*). Soit $\mathcal{N}$ l'ensemble des $n = |\mathcal{N}|$ nœuds où $||$ désigne le cardinal d'un ensemble et $\mathcal{A}$ l'ensemble des $m = |\mathcal{A}|$ arcs. Puisqu'un arc relie toujours exactement deux nœuds (mais deux nœuds ne sont pas nécessairement reliés), on a $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$. $\mathcal{G}(\mathcal{N}, \mathcal{A})$ définit donc un graphe.

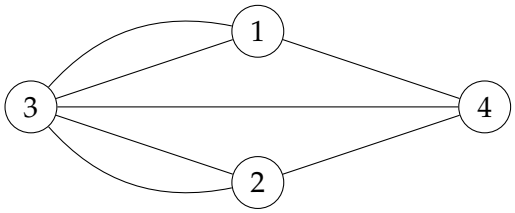


Figure 1.2 Les sept ponts de Königsberg représentés par un graphe. Les arcs représentent les ponts, les nœuds 1 et 2 la terre ferme et les nœuds 3 et 4 les îles

Être orienté ou pas

Lorsque les relations sont *symétriques*, le graphe est dit *non-orienté*. Formellement, un graphe est non-orienté lorsque $\forall (u, v) \in \mathcal{A}, (v, u) \in \mathcal{A}$. Un réseau informatique sera généralement non-orienté puisque deux ordinateurs peuvent communiquer dans les deux sens. À l'opposé, un réseau routier est orienté pour modéliser les rues à sens unique. Il est habituel (mais pas systématique) d'utiliser les termes *nœud* et *arc* pour les graphes orientés et *sommet* et *arête* pour les graphes non orientés.

Poids sur les arcs

Dans de nombreux problèmes, il est souhaitable de pouvoir qualifier une relation. Pour cela à chaque arc est associé un *poids* qui le décrit. Dans un réseau social il peut définir la nature de la relation (ami, famille, collègue) et dans un réseau routier la longueur d'une rue. Parfois le terme *coût* est utilisé.

En associant à chaque arc $\forall a \in \mathcal{A}$ un poids $c_a \in C$, on obtient un graphe $\mathcal{G}(\mathcal{N}, \mathcal{A}, C)$ qui est dit *valué*.

Même si dans les cas les plus courants les poids sont constants, il est possible d'utiliser des fonctions pour représenter les poids.

Successeurs et prédécesseurs

La fonction Γ^+ définit pour chaque nœud les *successeurs*, soit $\Gamma^+(u) = \{v \mid (u,v) \in \mathcal{A}\}$. Symétriquement on définit les *prédécesseurs* $\Gamma^-(u) = \{v \mid (v,u) \in \mathcal{A}\}$. Lorsque le graphe est symétrique les prédécesseurs et successeurs d'un nœud sont les mêmes. Plus généralement, deux nœuds sont dit *adjacents* lorsqu'ils sont reliés par un arc.

Degré et densité

Le *degré* d'un nœud est le nombre d'arcs adjacents à ce nœud et la *densité* d'un graphe est le rapport entre le nombre d'arcs sur le nombre de nœuds, soit $\frac{m}{n}$. On considère un réseau routier de type *Manhattan* où les rues se croisent toujours en un carrefour à quatre branches, si les rues sont généralement à double sens et en utilisant un graphe orienté, alors le degré moyen d'un nœud sera proche de 8, soit deux arcs par rue du carrefour. Dans ce même cas, la densité du graphe sera de l'ordre de 4.

1.1.2 Représentation

Pour représenter un graphe informatiquement, généralement deux modélisations sont utilisées.

Matrice d'adjacence

Une matrice $\mathcal{M}$ de dimension $n \times n$ représente un graphe de la manière suivante : $\mathcal{M}_{uv} = 1$ s'il existe un arc du nœud u au nœud v et 0 sinon. Plutôt que 1, il est possible d'utiliser le poids de l'arc pour marquer son existence. Si cette représentation permet de facilement tester l'existence d'un arc en $O(1)$, obtenir l'ensemble des successeurs d'un nœud est en $O(n)$. Enfin, la mémoire nécessaire pour stocker la matrice est en $O(n^2)$.

Liste d'adjacence

Dans cette représentation, un vecteur de n éléments contient pour chaque nœud la liste de ses successeurs. Cette représentation nécessite donc moins de mémoire qu'une matrice d'adjacence et obtenir la liste des successeurs d'un nœud est en $O(1)$. Lorsque le degré des nœuds est indépendant de la taille du graphe — comme dans un réseau routier —

alors la mémoire nécessaire est en $O(n)$. Cette représentation est donc préférée pour les graphes grands et peu denses.

1.2 Plus courts chemins simples

1.2.1 Définitions

Chemin

Un *chemin* (ou itinéraire ou route dans le contexte des transports) $r = \langle u_1, u_2, \ldots, u_l \rangle$ dans un graphe est une suite ordonnée de $l = |r|$ nœuds tels que deux nœuds successifs soient reliés par un arc $\forall i \in 1, \ldots, l-1, \quad (u_i, u_{i+1}) \in \mathcal{A}$. L'origine du chemin est appelé *source* et la destination *puits*.

Longueur d'un chemin

Dans un graphe valué par des valeurs réelles, la manière immédiate pour évaluer la longueur d'un chemin est de sommer le poids des arcs traversés :

$$c(c) = \sum_{i=1}^{l-1} c_{u_i, u_{i+1}}$$

Le *plus court chemin* entre deux nœuds est le chemin qui minimise la longueur parmi tous les chemins possibles entre ces deux nœuds. Cependant, au lieu de considérer la somme il est possible de considérer d'autres types d'objectifs tels que Min-Max (par exemple pour trouver un goulot d'étranglement) ou encore produit.

Arbre

Un *arbre* est une forme particulière de graphe où chaque nœud n'a au plus qu'un seul prédécesseur, soit $\forall u \in \mathcal{N}, \ |\Gamma^-(u)| \leq 1$. Un chemin est donc un cas particulier d'arbre.

Un *arbre couvrant* d'un graphe est un arbre qui contient tous les nœuds du graphe. Un arbre couvrant définit donc un chemin d'*un* nœud vers *tous* les autres.

1.2.2 Caractéristiques des algorithmes

De nombreux algorithmes ont été développés depuis les années 1950 pour calculer le plus court chemin dans un graphe. Ils se différencient par un certain nombre de caractéristiques :

- Poids acceptables : certains algorithmes n'acceptent que des arcs dont le poids est positif ;
- Chemins calculés : il existe des algorithmes qui calculent le plus court chemin de nœud à nœud, entre toutes les paires de nœuds ou encore d'un nœud vers tous les autres. Dans ce dernier cas, l'algorithme va générer un arbre couvrant ;
- Prise en compte d'informations externes : parfois l'utilisation d'une connaissance externe à la structure du graphe peut accélérer la recherche.

1.2.3 Principaux algorithmes

1.2.3.1 L'algorithme de Dijkstra

Initialement décrit dans Dijkstra [1], cet algorithme calcule le plus court chemin d'un nœud vers tous les autres et ne fonctionne qu'avec des arcs à poids positif.

Une étiquette (*label*) est associée à chaque nœud indiquant la plus courte distance trouvée entre le nœud et la source. Initialement cette étiquette vaut $+\infty$ pour tous les nœuds, sauf pour la source où elle sera initialement à 0.

Il s'agit d'un algorithme glouton, qui en partant de la source, sélectionne le nœud jamais exploré ayant l'étiquette la plus petite et met à jour les étiquettes de ses successeurs s'il y a une amélioration. Cette propriété peut être utilisée dans la recherche du plus court chemin entre deux points. En effet la recherche peut être arrêtée dès que le puits est le nœud de plus faible poids de la file `Q`. Le pseudo-code correspondant est indiqué dans **l'algorithme 1.1**.

À chaque itération, un nœud est retiré définitivement de la file `Q` et son poids est fixé définitivement. C'est pour cela qu'il s'agit d'un algorithme dit de *label-setting*, qui s'oppose aux algorithmes de *label-correcting*.

La complexité de cet algorithme dépend fortement des structures de données utilisées. Une implémentation naïve est en $O(n^2)$. L'utilisation de *tas de Fibonacci* pour la file de

priorité Q permet d'obtenir une complexité en $O(m + n \log n)$ Fredman,Tarjan [2]. Dans le cas particulier d'un réseau routier, puisque m est de l'ordre de n, la complexité est de seulement $O(n \log n)$.

```
1  FONCTION Dijkstra(source, nœuds, arcs, poids)
2      POUR TOUT u de nœuds FAIRE
3          dist[u] := +∞
4          pred[u] := indéfini
5      dist[source] := 0
6      Q := nœuds
7      TANT QUE Q est non vide FAIRE
8          u := nœud de Q mininisant dist[u]
9          Q := Q \ {u}
10         POUR TOUT v successeur de u FAIRE
11             si dist[v] > dist[u] + poids[u][v] alors
12                 dist[v] := dist[u] + poids[u][v]
13                 pred[v] := u
```

Algorithme 1.1 Algorithme de Dijkstra

1.2.3.2 L'algorithme de Bellman-Ford

L'algorithme 1.2 de Bellman-Ford calcule le plus court chemin d'un nœud vers tous les autres. Contrairement à l'algorithme de Dijkstra, à chaque itération *tous* les arcs sont examinés et pas seulement ceux adjacents au nœud le plus prometteur. Le chemin le plus court ayant au plus $n-1$ arcs (chemin comportant tous les nœuds du graphe), l'algorithme itère autant de fois.

Le principal avantage de cet algorithme sur celui de Dijkstra est qu'il est possible de détecter des circuits absorbants. Un circuit absorbant est une boucle dont le poids total est négatif. Le plus court chemin bouclerait donc indéfiniment le long de ce circuit. Cet algorithme permet donc d'utiliser des arcs avec des poids négatifs. Cependant, si un tel circuit absorbant est détecté, l'algorithme s'arrête. Il ne permet pas de résoudre le problème du plus court chemin avec des arcs de poids négatifs sans circuit.

Comme le plus court chemin jusqu'à un nœud peut être corrigé à chaque itération, on parle d'algorithme de type *label-correcting*.

La complexité de cet algorithme est en $O(n \cdot m)$, soit dans le cadre d'un réseau routier $O(n^2)$.

```
1  FONCTION Bellman-Ford(source, nœuds, arcs, poids)
2      POUR TOUT u de nœuds FAIRE
3          dist[u] := +∞
4          pred[u] := indéfini
5      dist[source] := 0
6      RÉPÉTER |nœuds| FOIS
7          POUR TOUT (u,v) de arcs FAIRE
8              si dist[v] > dist[u] + poids[u][v] alors
9                  dist[v] := dist[u] + poids[u][v]
10                 pred[v] := u
11     POUR TOUT (u,v) de arcs FAIRE
12         si dist[v] < dist[u] + poids[u][v] alors
13             Cycle absorbant détecté
```

Algorithme 1.2 Algorithme de Bellman-Ford

1.2.3.3 Améliorations pour le plus court chemin de point à point

Lorsque le plus court chemin recherché est entre deux nœuds, il est possible d'utiliser cette propriété pour réduire les nœuds explorés et donc réduire le temps d'exécution. Cependant aucune des approches ne permet d'améliorer la complexité dans le pire cas.

L'algorithme de Dijkstra bidirectionnel consiste à effectuer en même temps deux calculs d'itinéraire, l'un en partant de la source et l'autre du puits. Lorsque les deux recherches ont toutes les deux exploré un même nœud, alors l'exploration est arrêtée et le plus court chemin est la concaténation des deux chemins trouvés vers ce nœud en commun.

Une approche bien connue est l'algorithme A* Hart, [3] qui utilise une métrique permettant d'estimer la distance jusqu'au puits. Ainsi pour le plus court chemin dans un réseau routier, il est habituel d'utiliser la distance euclidienne. Si cette heuristique est la fonction `h`, l'algorithme est obtenu en remplaçant la ligne 8 de l'algorithme de Dijkstra par `u := nœud de Q minimisant (dist[u] + h(u))`. Si cette heuristique sous-estime la distance réelle, alors l'algorithme est optimal. Il est cependant possible de tolérer une légère sur-estimation afin de réduire le temps d'exploration au risque de ne pas être optimal.

1.2.3.4 Approches haute performance

Ces dernières années, une course aux meilleures performances a eu lieu avec des résultats assez spectaculaires. Le calcul du plus court chemin à travers les États-Unis (24 millions de nœuds et 30 millions d'arcs) est effectué en moins de 1 ms sur un ordinateur de bureau. Les nouvelles approches peuvent être regroupées en trois grandes classes.

Approches hiérarchiques

Ces approches définissent un ordre sur les nœuds. Lors de la recherche de l'itinéraire, il n'est pas toujours possible de passer d'un nœud à un autre, même lorsqu'un arc existe. Le *Highway-Node Routing (HNR)* Schultes [4] regroupe les nœuds en classes hiérarchisées. La recherche est bi-directionnelle et n'autorise que de rester dans la même classe ou d'aller dans une classe supérieure. Le rajout de nouveaux arcs de raccourci (*shortcuts*) permet de garantir de trouver effectivement le plus court chemin. Le nombre de classes ainsi que la répartition des nœuds dans ces différentes classes est une approche heuristique. L'heuristique la plus utilisée consiste à minimiser le nombre d'arcs de raccourci qui sont rajoutés.

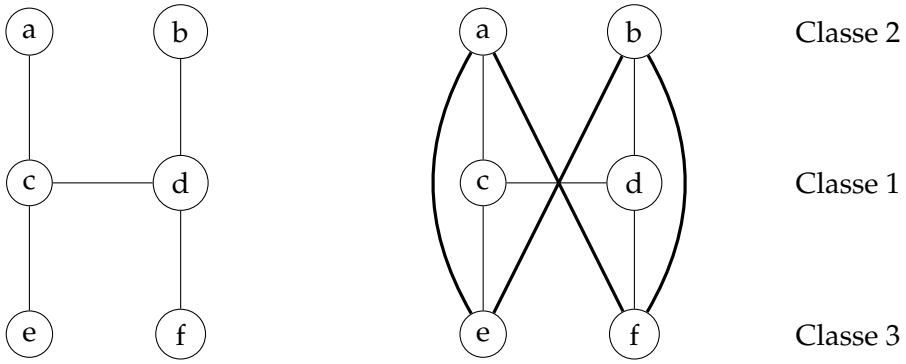


Figure 1.3 Exemple de classifications de nœuds pour le Highway Node Routing avec trois classes. Les nœuds c et d sont donc hiérarchiquement plus bas que les nœuds a et b qui sont plus bas que les nœuds e et f

La **figure 1.3** présente un exemple avec six nœuds et trois classes. Le graphe à gauche représente le réseau routier avant le rajout des raccourcis. Les raccourcis dans le graphe de droite sont représentés par des arcs plus épais. Ainsi grâce à ces arcs, il est possible de passer d'un nœud de la classe 2 à la classe 3 sans passer par la couche 1 plus basse dans la hiérarchie.

Une variante très simple et plus efficace est *Contraction Hierarchies* Geisberger, [5] qui considère que chaque nœud est seul dans une classe. L'approche *Highway Hierarchies* Sanders,Schultes [6] classe non pas les nœuds, mais les arcs. En étant éloignés du nœud d'arrivé et de départ, les arcs les plus importants sont privilégiés.

L'optimalité de ces approches n'est pas évidente au premier abord. Par soucis de simplicité nous n'en présentons pas ici la démonstration. Cependant dans le cadre de l'optimisation multiobjectif, nous avons adapté les *contraction hierarchies* et présentons dans le chapitre 5 une preuve d'optimalité.

Calcul en amont de certaines distances

L'approche ALT Goldberg,Harrelson [7] sélectionne un nombre réduit (de l'ordre de la dizaine) de nœuds (appelés *landmarks*) qui serviront de repère. Ensuite l'algorithme calcule le plus court chemin des repères vers tous les autres nœuds. Les repères peuvent être choisis aléatoirement donnant des résultats acceptables d'après les auteurs. Cependant ils conseillent dans le cadre de réseaux routiers l'approche suivante : choisir un premier repère aléatoirement, puis choisir le nœud le plus éloigné de ce premier nœud. Les repères suivants seront choisis de manière à maximiser la distance au repère le plus proche. Ces données permettent d'améliorer significativement l'heuristique de distance dans une recherche de type A*.

Le *Transit-Node Routing* Bast, [8] part de l'observation « humaine » que certains nœuds sont utilisés pour de nombreuses routes et d'autres très rarement (presque tout voyage passera par le périphérique, quelque soit le point de départ et d'arrivée). En calculant à l'avance le plus court chemin entre ces nœuds importants, le plus court chemin peut être obtenu très rapidement.

Marquage d'arcs

L'approche *arc-flags* (parfois *edge-flags*) Hilger, [9] découpe le graphe en régions. Le découpage le plus simple est obtenu en quadrillant l'espace. Cependant les auteurs présentent d'autres découpages pour équilibrer le nombre d'arcs dans chaque région. Pour couvrir l'Allemagne, les auteurs considèrent une trentaine de régions.

Sur chaque arc, pour chaque région, un marqueur indique si le plus court chemin vers cette région passe par l'arc. Ainsi lors de la recherche d'itinéraire, certains arcs pourront

être ignorés et la zone de recherche est orientée vers la destination. Ainsi la recherche d'un itinéraire à travers les États-Unis est de l'ordre de quelques millisecondes.

Ce découpage est illustré dans la **figure 1.4** avec trois régions. Sur chaque arc un triplet de binaires indique si l'arc permet d'atteindre chaque région. Ainsi l'arc (c,d) ne permet d'atteindre aucune région (sauf les nœuds de la région courante) et l'arc (a,e) ne permet dans aucun cas d'atteindre la région B.

Le découpage peut être hiérarchique afin de ne pas devoir mémoriser trop d'informations sur chaque arc et pouvoir continuer à utiliser cette approche après avoir atteint la région.

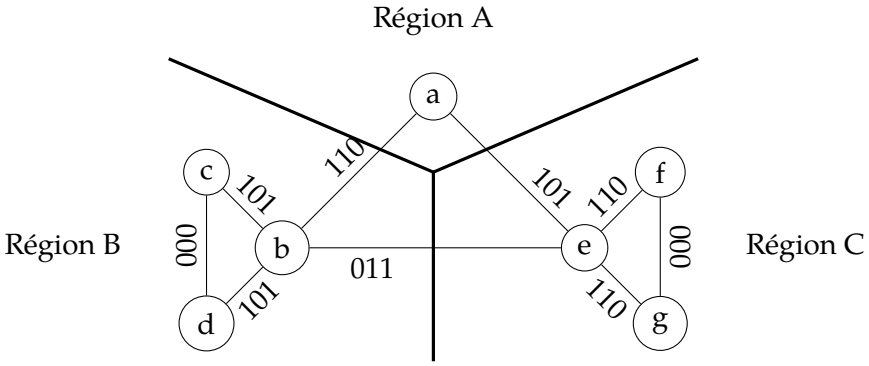


Figure 1.4 Exemple de partitionnement du graphe pour l'approche *arc-flags*. Le premier bit à 1 veut dire qu'il existe un chemin qui emprunte l'arc pour atteindre la région A, le deuxième bit la région B et le troisième bit la région C

Toutes ces approches ont été combinées entre elles, donnant lieu à de nouvelles techniques encore plus performantes. Aucune approche ne dépasse significativement une autre, car il s'agit de compromis entre temps de calcul, temps de pré-traitement et surcoût en mémoire. Ces combinaisons sont présentées dans Bauer, [10], Delling, [11].

Cependant, les performances de ces algorithmes n'ont été mesurées que expérimentalement. Les auteurs ne donnent aucune preuve de leurs performances théoriques ou sur des graphes qui ne sont pas routiers (un réseau de télécommunications par exemple). Cette lacune est comblée dans Abraham, [12] qui montre qu'il est possible de prouver théoriquement les performances en pré-traitement, consommation mémoire et temps de calcul pour la majorité de ces algorithmes.

1.3 Plus court chemin avec coûts variables

Les algorithmes vus précédemment supposent que tous les coûts sont constants et additifs. Ce modèle présente cependant rapidement des limites : s'il est possible de considérer

que la longueur d'un arc routier est constante dans le temps, le temps de parcours peut varier, par exemple en fonction des conditions de trafic. La situation se complique d'autant plus si l'on désire prendre en compte des grilles horaires (cas des transports en commun par exemple).

1.3.1 Formalisation

Nous nous restreignons au cas où les coûts dépendent du temps. Nous verrons qu'il existe deux familles distinctes de problèmes selon que la fonction objectif à minimiser est le temps (chemin *le plus rapide*) ou une fonction de coût arbitraire (chemin *de moindre coût*). De plus nous considérons que l'instant de départ t_0 est fixé à l'avance.

Afin de généraliser, nous remplaçons l'ensemble des coûts $\mathcal{C}$ par deux ensembles de fonctions $\mathcal{F}^t$ et $\mathcal{F}^c$ qui définissent respectivement pour chaque arc (u, v) et pour un instant t donné :

- l'instant d'arrivée en v en partant de u à l'instant t ;
- le coût de l'arc (u, v) en partant de u à l'instant t.

Étant donné un chemin $r = \langle u_1, \ldots, u_l \rangle$, on peut ainsi définir pour chaque nœud u_i de ce chemin l'instant d'arrivée t_i à ce nœud :

$$t_i = f^t_{u_{i-1},u_i} \circ f^t_{u_{i-2},u_{i-1}} \circ \ldots \circ f^t_{u_1,u_2}(t_0)$$

Connaissant l'instant d'arrivée à chaque nœud, il est alors possible de calculer le coût du chemin :

$$c(r) = \sum_{i=1}^{l-1} f^u_{u_i,u_{i+1}}(t_{i-1})$$

1.3.2 Chemin le plus rapide sans contrainte

Lorsque l'objectif à minimiser est le temps et qu'il n'existe aucune contrainte sur le parcours d'un arc (par opposition au cas où l'on doit respecter une grille horaire) il est possible d'utiliser l'algorithme de Dijkstra à peine modifié comme le montre **l'algorithme 1.3**. Ce cas s'applique par exemple pour calculer le chemin le plus rapide dans un réseau routier en prenant en compte l'évolution des conditions de trafic dans le temps.

Cette approche a été proposée initialement dans Dreyfus [13], sans pour autant fournir de preuve quant à son optimalité. Les conditions exactes d'optimalité sont définies dans Gondran,Minoux [14]. Une condition suffisante pour que l'algorithme de Dijkstra généralisé soit optimal est que les fonctions soient croissantes et qu'elles tendent vers l'infini lorsque le temps tend vers l'infini.

```
1  FONCTION Dijkstra_généralisé(source, nœuds, arcs, ft)
2      POUR TOUT u de nœuds FAIRE
3          t[u] := +∞
4          pred[u] := indéfini
5      t[source] := t0
6      Q := nœuds
7      TANT QUE Q est non vide FAIRE
8          u := nœud de Q minimisant t[u]
9          Q := Q \ {u}
10         POUR TOUT v successeur de u FAIRE
11             si t[v] < ft[u][v](t[u]) alors
12                 t[v] := ft[u][v](t[u])
13                 pred[v] := u
```

Algorithme 1.3 Algorithme de Dijkstra généralisé (dépendant du temps)

Amélioration des performances

Pour le plus court chemin avec des poids statiques, de nombreuses techniques permettent d'améliorer le temps d'exécution. Cependant, l'adaptation de ces techniques usuelles, telles que l'utilisation de bornes inférieures ou la recherche bidirectionnelle, est loin d'être immédiate. En effet la recherche bidirectionnelle nécessiterait de connaître à l'avance l'instant d'arrivée au nœud d'arrivée. Cependant, d'importants efforts ont été faits dans ce domaine et les techniques haute performance présentées dans le cadre de coûts statiques ont été adaptées avec succès Nannicini, [15], Delling [16], Batz, [17].

Variantes

Dans le cadre de sa thèse, Hizem [18] s'intéresse à des variantes du chemin le plus rapide. Il s'agit du problème d'interception d'un mobile dans un graphe qui consiste à rejoindre le plus rapidement un objet qui se déplace selon un trajet connu dans le graphe. La

deuxième variante à laquelle il s'est intéressé est le problème du plus court chemin dans un graphe dynamique avec intervalles. Dans le problème, les coûts sont représentés par des intervalles afin de modéliser l'incertitude sur le poids des arcs. Par exemple si le coût est de 10 avec une incertitude de plus ou moins 10%, alors le coût sera dans l'intervalle [9 ;11].

1.3.3 Prise en compte des grilles horaires

Pour modéliser un réseau de transports en commun, ou un réseau routier dont on voudrait prendre en compte les feux de circulation, il est indispensable de pouvoir modéliser l'attente à un nœud. Il n'est donc pas possible d'utiliser directement l'approche vue précédemment.

1.3.3.1 Contrainte FIFO

La contrainte FIFO *First In, First Out* est particulièrement importante. Cette contrainte stipule qu'il est nécessaire qu'emprunter un arc (u, v) au plus tôt garantisse le fait d'arriver en v au plus tôt.

Un exemple où cette contrainte n'est pas respectée est celui d'un bus dépassant un autre entre deux arrêts. Il aurait été plus optimal d'attendre le deuxième bus pour arriver plus tôt à destination. Il s'agit d'un cas où la fonction de coût associée à l'arc n'est pas croissante partout.

Afin de pouvoir prendre en compte le temps d'attente et respecter la contrainte FIFO, plusieurs approches sont généralement employées. Historiquement, les premières approches utilisées se basent sur une approche en deux passes. Une première étude sur le graphe statique permet de sélectionner un certain nombre d'itinéraires potentiels. Une deuxième passe permet d'examiner les horaires disponibles afin de trouver le meilleur itinéraire. D'après Muller-Hannemann, [19], c'est cette approche qui est utilisée pour le système *HAFAS* qui est utilisé entre autres par les chemins de fer allemands et qui est capable de calculer des itinéraires à travers toute l'Europe.

Cette approche étant heuristique et plutôt compliquée à mettre en œuvre, deux autres approches sont habituellement employées permettant une modélisation et une résolution plus simple. La première consiste à transformer le graphe pour se ramener à une situation où les coûts sont constants (modèle *time-expanded*). La deuxième approche modifie les fonctions de coût afin d'intégrer les temps d'attente (modèle *time-dependent*).

1.3.3.2 Modèle *Time-expanded*

Habituellement dans un graphe, un nœud correspond à une position géographique. Ce modèle dédouble les nœuds et les associe à un instant précis. Le graphe généré est appelé graphe *espace-temps* (*time-space graph*). Dans ce graphe chaque nœud est en fait un couple entre un nœud du graphe initial et un instant. Les arcs définissent les possibilités de passer d'un nœud à un autre dans l'intervalle de temps défini par les deux instants associés aux nœuds. Le poids associé à chaque arc est la différence de temps entre les instants correspondant aux deux nœuds.

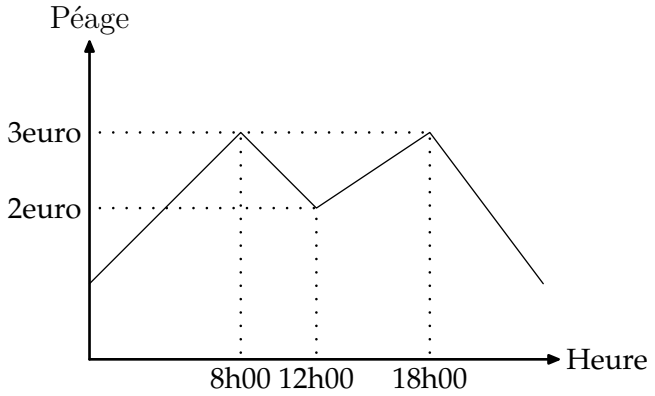


Figure 1.5 Exemple de graphe espace-temps . Pour chaque moment, les nœuds sont démultipliés pour modéliser la possibilité de passer d'un nœud à un autre à un instant donné

Les instants peuvent être définis de deux manières : en discrétisant le temps ou en définissant un instant par évènement.

Discrétiser le temps

Le temps est divisé en q intervalles de temps. Cette approche a l'avantage de la simplicité. Cependant si les intervalles de temps sont trop petits, la taille du graphe augmente significativement. À l'opposé, lorsque l'intervalle de temps est trop grand, le résultat peut être très imprécis.

La **figure 1.5** représente un réseau ferroviaire dans la partie gauche. Le temps de parcours entre deux nœuds est indiqué sur les arcs. Un train part de chaque nœud toutes les 15 minutes. Le graphe de la partie droite correspond au graphe espace-temps représentant le réseau ferroviaire sur cinq intervalles de temps.

Cette approche est proposée dans Pallottino,Scutella [20]. L'algorithme proposé par les auteurs *Chrono-SPT* permet de gérer implicitement le graphe espace-temps permettant ainsi de réduire la consommation en mémoire. L'algorithme fonctionne en $O(mq)$ où m est le nombre d'arcs et q le nombre d'intervalles considérés.

Un instant par « évènement »

Un évènement d'un nœud est par exemple l'entrée en gare d'un train. Ainsi un nœud sera dédoublé en autant de nœuds qu'il y a d'évènements associés à ce nœud. Cela permet d'avoir un graphe de moindre taille pour des nœuds à faible activité et surtout d'être parfaitement optimal.

Avec un tel modèle, il devient aisé de modéliser une contrainte, telle que laisser une certaine marge pour un changement de train : il suffit de ne pas relier deux nœuds si les instants associés sont considérés comme trop proches.

Puisque le coût des arcs est constant, il est possible d'utiliser un algorithme de Dijkstra traditionnel. La taille supplémentaire du graphe est compensée par le fait que le graphe a une densité très faible et que de plus il est acyclique.

Ce modèle a été repris dans Bast, [21] qui est probablement à ce jour l'approche la plus performante pour le calcul d'itinéraires en transport en communs. La représentation des arrêts et des évènement est assez immédiate. Afin d'augmenter les performances, les auteurs calculent à l'avance des *transfer pattern* qui permettent d'avoir les correspondances à utiliser entre deux arrêts indépendamment de l'heure. Au moment du calcul de l'itinéraire, seuls certains tronçons sont explorés permettant un calcul particulièrement véloce tout en permettant une optimisation à deux objectifs. Cependant, calculer toutes les correspondances entre deux points étant trop long, des approximations légères sont nécessaires risquant de générer un résultat non optimal. Cet algorithme est appliqué à grande échelle par Google sur de très grandes échelles telles que toute la Suisse.

1.3.3.3 Modèle *Time-dependent*

Ce modèle modifie les fonctions de coût des arcs : plutôt que de correspondre à la durée de parcours, les fonctions renvoient l'instant d'arrivée au plus tôt au nœud cible. Ainsi la **figure 1.6** montre la fonction d'un train partant toutes les 15 minutes pour un trajet durant 15 minutes. Une personne arrivant à l'arrêt à 7h46 ou à 7h59 arrivera dans les deux cas à destination à 8h15.

La **figure 1.7** représente la fonction pour un tronçon routier. À minuit un automobiliste ne mettra que 10 minutes pour le parcourir, mais mettra 20 minutes à 8h00 du matin à cause de la circulation.

La taille du graphe reste donc constante, mais nécessite l'utilisation de l'algorithme de Dijkstra modifié. Toutes ces fonctions sont non-décroissantes et tendent vers l'infini. De ce fait un algorithme de Dijkstra dépendant du temps reste optimal.

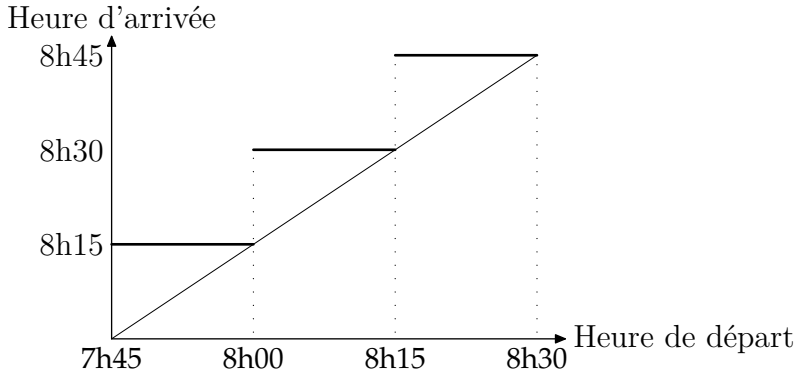


Figure 1.6 Exemple de fonction de coût sur un arc de transports en commun dans un modèle *time-dependent*

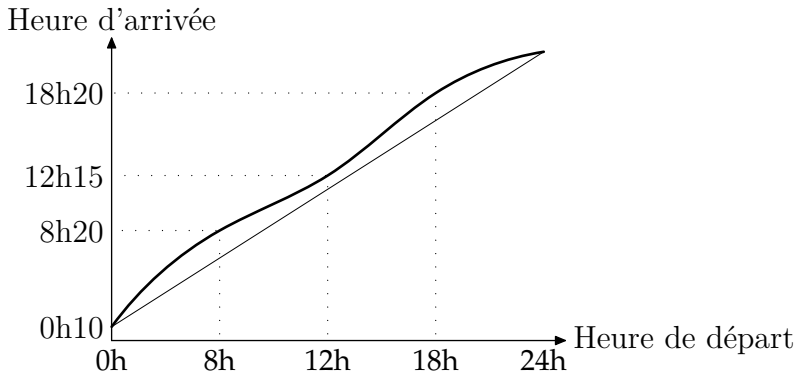


Figure 1.7 Exemple de fonction de coût sur un arc routier dans un modèle *time-dependent*. La courbe de gauche est la fonction de coût à proprement parler et la courbe de gauche indique le temps de parcours de l'arc routier en fonction de l'heure

Pouvoir gérer les temps de changement est plus délicat. Il est en effet nécessaire de dédoubler les nœuds afin de pouvoir rajouter un arc modélisant le temps de changement nécessaire comme cela est montré dans la **figure 1.8**.

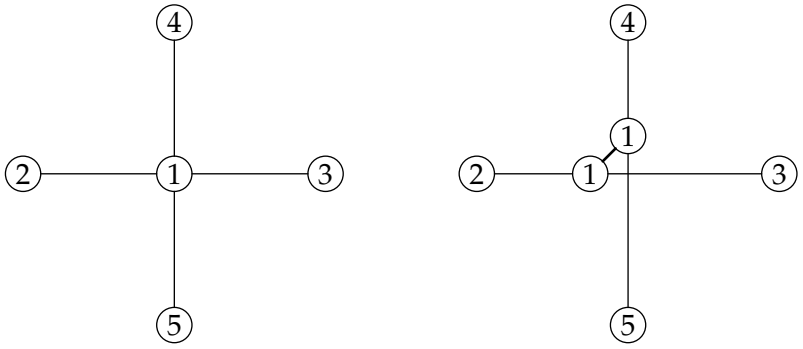


Figure 1.8 Exemple de gestion du temps de changement : (2,1,3) et (4,1,5) représente respectivement deux lignes de métro ayant la station 1 en commun

Afin de permettre d'utiliser les *contraction hierarchies*, dans Geisberger [22], les auteurs séparent la topologie du graphe des fonctions de coût. Cela permet d'appliquer une contraction classique sur le graphe initial. Les auteurs proposent leur propre algorithme pour pouvoir prendre en compte les cas non-FIFO liés à cette contraction. Il s'agit probablement de l'approche la plus performante actuellement existante en termes de temps de calcul dans le cas très précis de l'itinéraire multimodal le plus rapide.

1.3.3.4 Comparaison des deux modèles

Les deux modèles sont comparés expérimentalement dans Pyrga, [23]. Il semblerait que ce soit actuellement le seul comparatif expérimental des deux approches. Les auteurs notent un net avantage pour le modèle *Time-dependent* d'un point de vue performance. Cependant le modèle *Time-expanded* se révèle être plus commode pour une modélisation plus fine (par exemple la prise en compte du temps pour changer de quai). De plus l'écart de performance se réduit avec la complexification du modèle.

Afin d'améliorer les performances, plusieurs techniques d'accélération du modèle *Time-expanded* sont présentées dans Delling, [24]. Même si l'amélioration est d'un facteur 50 par rapport à un algorithme tel que celui de Dijkstra, ces améliorations sont faibles comparées à celles obtenues avec un graphe en appliquant des améliorations sur le modèle *Time-dependent* telles que les *contraction hierarchies*.

1.3.4 Chemin de moindre coût

Lorsque la fonction de coût à optimiser n'est pas le temps mais qu'elle dépend du temps, il n'est plus possible d'utiliser l'algorithme de Dijkstra. Il est peu probable qu'il existe un

algorithme en temps polynomial : ce problème est en effet NP-difficile puisqu'il permet de résoudre le problème de plus court chemin avec des contraintes de ressources.

Le premier article (d'après leurs auteurs) qui traite de ce problème est Orda,Rom [25]. Les auteurs montrent que le chemin de moindre coût peut être de longueur infinie. Un algorithme est proposé pour résoudre ce problème avec des conditions particulières sur les fonctions de coût pour que le chemin optimal soit de longueur finie. L'exécution de l'algorithme s'arrêtera , mais rien ne garantit qu'il se termine en temps polynomial.

La différence entre les deux types d'objectifs est développée dans Ahuja, [26] où les auteurs étudient le chemin le plus rapide et celui de moindre coût dans un réseau routier prenant en compte les feux et démontrent la différence de complexité.

Cependant dans certains cas, le problème de moindre coût a une complexité polynomiale. Il s'agit des graphes dit *cost-consistent*. Cette contrainte supplémentaire stipule que quitter un nœud plus tôt ne coûte pas plus cher que le quitter plus tard. Cette notion est présentée dans Pallottino,Scutella [20]. Les auteurs présentent également l'adaptation de leur algorithme *Chrono-SPT* pour calculer le chemin de moindre coût sur de tels graphes.

Un résultat intéressant est celui présenté dans Chabini [27]. Les auteurs présentent un algorithme très simple pour calculer le chemin de moindre coût de *tous* les nœuds vers *une* destination à *tous* les intervalles de temps. Le temps est discrétisé et le temps de parcours des arcs doit être entier de manière à correspondre exactement à un intervalle de temps. L'algorithme part du dernier instant, puis remonte à chaque itération d'un intervalle de temps et pour chaque arc étudie si à cet instant il est possible d'améliorer le coût du nœud cible de l'arc. La complexité de l'algorithme est en $O(n \log n + nM + mM)$ avec n nœuds, m arcs et M intervalles de temps. Cette approche est performante comme le prouvent leurs expériences (moins d'une seconde pour 3 000 nœuds, 9 000 arcs et 90 intervalles de temps). Cependant, cette approche n'est envisageable que si la durée de parcours sur un arc est nettement plus grande que l'intervalle de temps considéré. Ainsi une voiture mettra 10 secondes entre deux carrefours, ce qui nécessite des intervalles de temps de l'ordre de la seconde.

Les auteurs ont prouvé qu'il était impossible d'améliorer cette complexité. En conséquence, dans le but de réduire les temps d'exécution, ils ont proposé une variante parallélisée dans Chabini,Ganugapati [28] afin d'obtenir le résultat plus rapidement.

1.4 Optimisation multiobjectif

L'optimisation multiobjectif (aussi dite *multicritère* dans certaines communautés) consiste

à optimiser plusieurs objectifs contradictoires. Ainsi vouloir l'itinéraire le plus rapide et le moins cher est généralement incompatible. En effet l'itinéraire le moins cher serait de tout faire à pied, tandis que le plus rapide pourrait être de prendre un hélicoptère.

Afin de gérer les objectifs contradictoires, un décideur (c'est-à-dire l'utilisateur du système) est obligé d'intervenir. Le moment auquel le décideur intervient définit les trois grandes familles d'algorithmes de résolution :

- *a priori* : il exprime ses préférences avant la résolution ;
- *interactive* : il prend part au processus de résolution ;
- *a posteriori* : il choisit une solution parmi un ensemble de solutions potentielles.

1.4.1 Formalisme et notations

Nous considérons par la suite, sans perdre en généralité, que nous souhaitons minimiser tous les objectifs. Afin de comparer deux solutions, il est possible de définir un ordre partiel habituellement appelé *dominance* au sens de *Pareto*. Une solution *domine strictement* une autre solution, si elle est meilleure ou égale sur tous les objectifs et strictement meilleure sur au moins un objectif.

Nous utilisons les notations utilisées dans Coello, [29], même si les auteurs ne s'intéressent qu'à une branche particulière de l'optimisation multiobjectif. Pour une vision plus générale, une autre référence est Ehrgott [30].

Une solution est un vecteur $\mathbf{x} \in \mathbb{R}^{\mathbf{n}}$ de n variables de décision :

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

L'ensemble des variables de décision est appelé l'espace de décision et est noté Ω.

Dans un problème d'optimisation à k objectifs, on définit $f_1(\mathbf{x})$, $f_2(\mathbf{x})$,..., $f_k(\mathbf{x})$ formant ainsi un vecteur objectif $\mathbf{f} : \mathbb{R}^{\mathbf{n}} \rightarrow \mathbb{R}^{\mathbf{k}}$:

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_k \end{bmatrix}$$

Un vecteur de l'espace des objectifs $\mathbf{u} = (u_1, \ldots, u_k)$ *domine* un autre vecteur $\mathbf{v} = (v_1, \ldots, v_k)$ (noté $\mathbf{u} \prec \mathbf{v}$) si et seulement si :

$$\forall i \in \{1, \ldots, k\},\ u_i \leq v_i,\ \exists i \in \{1, \ldots, k\}\ u_i < v_i$$

Un *ensemble Pareto-optimal* est un ensemble dans l'espace de décision Ω dont aucun élément n'en domine un autre dans l'espace des objectifs :

$$\mathcal{P}^* := \{\mathbf{x} \in \Omega \mid \nexists \mathbf{x}' \in \Omega,\ \mathbf{f}(\mathbf{x}) \prec \mathbf{f}(\mathbf{x}')\}$$

Le front de Pareto $\mathcal{PF}^*$ d'un problème est l'ensemble dans l'espace des objectifs tel que :

$$\mathcal{PF}^* := \{\mathbf{u} = \mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{P}^*\}$$

L'ensemble des solutions qui ne sont pas dominées par une autre solution, est appelé *front de Pareto*. Quelque soit la technique d'optimisation, la solution trouvée doit appartenir à ce front. En effet, toute solution dont au moins un des objectifs peut être amélioré sans pour autant sacrifier un autre objectif, sera préférée.

La **figure 1.9** montre le front de Pareto formé par les points A, B, C. Le point D est dominé par B. En revanche A et D sont deux solutions non comparables.

Les applications de l'optimisation sont considérables comme le montre l'état de l'art de référence Ehrgott,Gandibleux [31]. Les applications plus spécifiquement orientées vers le problème du plus court chemin sont développées dans Raith,Ehrgott [32] ou encore dans Clímaco,Pascoal [33].

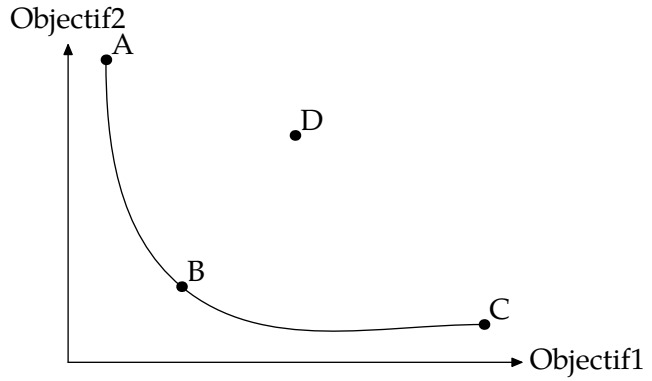


Figure 1.9 Exemple de front de Pareto avec deux objectifs à minimiser. Les points A, B et C sont Pareto optimal, tandis que le point D est strictement dominé par les trois autres

1.4.2 Complexité

Le nombre d'éléments dans le front de Pareto peut devenir exponentiel sur certains problèmes. De ce fait, calculer cet ensemble est au moins exponentiel en temps. Cependant, dans un certain nombre de cas, la taille de ce front est polynomiale, rendant le problème plus simple, comme celui décrit par Muller-Hannemann,Weihe [34].

Les auteurs s'intéressent tout particulièrement au transport ferroviaire de passagers avec comme objectifs la minimisation du temps de parcours, le coût et le nombre de changements.

Les arcs du réseau correspondent à chaque couple gare de départ – gare d'arrivée pour chaque train. Ces arcs sont regroupés en *classes* disjointes correspondant à chaque catégorie de train (grande vitesse, longue distance, régional, etc.).

Dans le cadre d'optimisation bicritère, il peut être réaliste de considérer qu'une valeur d'un arc est le produit d'une constante par une constante pour une classe donnée (la durée est la distance multipliée par la vitesse et la vitesse est considérée constante entre deux classes de train). Il est donc possible de définir un ordre total entre les différentes classes. À partir de cet ordre nous pouvons définir la monotonie d'un trajet (en France un trajet TER-Corail-TGV sera monotone mais pas TGV-métro-TGV). Les auteurs décrivent un chemin comme *bitonic* un chemin pouvant être séparé en deux parties monotones, la première croissante et la deuxième décroissante. En effet, dans de nombreux cas, on utilise des moyens de transport lents autour du lieu de départ et d'arrivée et un moyen rapide au milieu. Les auteurs démontrent que le nombre de chemins *bitonics* est en $O(|V|^{2b-2})$ avec b classes et $|V|$ nœuds.

Des expériences ont été faites à partir de 1 000 itinéraires issus de requêtes utilisateur réelles à l'échelle de toute l'Allemagne, sur des instances n'étant pas découpées en classes et sur des instances aléatoires mais organisées en classes. Les observations montrent qu'effectivement le nombre d'éléments dans le front de Pareto est significativement plus faible avec les conditions citées précédemment (en moyenne de l'ordre de 3,5). De plus, la part de chemins *bitonic* parmi tous les chemins trouvés est de 84% en moyenne. Cela est facilement compréhensible, puisque les dessertes plus fines, souvent nécessaires pour commencer et finir un voyage, sont effectuées par des modes de transport plus lents.

1.4.3 Approches *a priori*

Ces approches ont été les premières utilisées. Elles consistent à construire une fonction

objectif qui englobe tous les objectifs afin de se ramener à une optimisation à un seul objectif, permettant ainsi d'utiliser des algorithmes mieux connus.

Il est à noter que dans le cas particulier du plus court chemin multiobjectif dépendant du temps, cette approche ne simplifie pas nécessairement le problème, puisque nous avons vu précédemment que si l'objectif à optimiser n'est pas le temps, alors le problème devient généralement NP-difficile.

Nous présentons trois techniques qui peuvent être utilisées pour une approche *a priori*.

Ordre lexicographique

Le décideur classe les objectifs par ordre de préférence. Ensuite, l'algorithme va exécuter plusieurs optimisations, en commençant par celui de plus grande importance. Cette approche est rarement utilisée. En effet elle se focalise trop sur des solutions extrêmes, alors que généralement le décideur cherche une solution de compromis.

Cette approche est cependant pertinente dans un calcul d'itinéraire multimodal : en effet, souvent une solution impliquant le moins de changement sera préférée, même si cela implique un temps de parcours plus long. Cette approche a été utilisée dans Pyrga, [23].

Linéarisation des objectifs

La somme pondérée de chaque objectif est l'approche la plus simple. Celle-ci a du sens si chaque objectif peut être ramené à une même unité de mesure (dans une entreprise, un retard d'une heure peut être mesuré en tant que perte d'argent). Cependant ce n'est pas toujours possible : par exemple si l'on désire optimiser le confort, rester debout est parfaitement acceptable pour un trajet de quinze minutes, mais intolérable pour un trajet plus long. Comment dans ce cas linéariser le coût et le confort ?

Outre le problème de modélisation, la linéarisation ne permet de trouver que des solutions sur l'enveloppe convexe du front de Pareto. Ainsi une solution qui aurait été la préférée du décideur risque d'être écartée.

C'est dans cette optique que la norme de Tchebycheff peut être utilisée. Entre deux solutions $\mathbf{u}$ et $\mathbf{v}$ et un vecteurs de poids $\mathbf{w}$, la norme est $\max_i\{w_i|p_i - r_i|\}$. L'expert peut alors définir une solution idéale $\mathbf{u}$ (ou alors cette solution est déduite en optimisant individuellement chaque objectif). La solution optimale est donc la solution minimisant la norme de Tchebycheff à la solution idéale. Dans Galand,Perny [35] plusieurs approches basées

sur A* sont présentées pour trouver la meilleure solution de compromis définie par une norme de Tchebycheff.

Intégrale de Choquet

L'intégrale de Choquet est un moyen de modéliser les préférences du décideur plus finement que par la simple linéarisation. Il est en particulier possible de définir des préférences entre deux solutions, plutôt que simplement associer une note à chaque solution. Elle se révèle particulièrement utile pour modéliser des préférences hétérogènes (coût et confort par exemple). Une introduction générale à cette approche est disponible dans Grabish [36].

Cette intégrale a été utilisée dans le cadre du plus court chemin dans Fouchal, [37] ou encore dans Galand, [38].

1.4.4 Approches interactives

Les approches interactives consistent à proposer une solution au décideur qui choisira ensuite sur quels objectifs il souhaiterait améliorer la solution. Le système propose alors une nouvelle solution. Le processus est répété jusqu'à ce que le décideur soit satisfait. Cette approche a été appliquée au problème de plus court chemin multiobjectif dans Gabrel,Vanderpooten [39] ou encore Galand [40].

Il faut cependant noter qu'il ne s'agit pas d'une approche *a priori* pour laquelle l'utilisateur change à chaque itération le paramétrage. Afin d'améliorer les performances, les approches interactives sont capables d'exploiter les résultats obtenus dans le calcul précédent. Elles sont donc surtout adaptées dans le cas où le temps de calcul est particulièrement important.

1.4.5 Approches *a posteriori*

Ces techniques visent à obtenir le front de Pareto le plus complet possible. L'ensemble des solutions est alors présenté au décideur qui choisira celle qu'il considère comme la plus adaptée. Cette approche a l'avantage de n'éliminer aucune solution qui aurait pu intéresser le décideur. La contrepartie est que la représentation de l'ensemble du front de Pareto est aisée avec deux objectifs, mais devient problématique au delà. De plus, le décideur souhaite généralement devoir uniquement choisir entre quelques solutions et non entre plusieurs milliers, d'autant plus que de nombreuses solutions sont très similaires.

1.4.5.1 Métaheuristiques

L'adaptation de métaheuristiques à une approche multiobjectif (très largement représentés par la *evolutionary multicriterion optimization*) concentre la majorité des efforts de recherche dans le domaine de l'optimisation multiobjectif, en particulier en se basant sur les algorithmes génétiques.

Algorithmes génétiques

Parmi les approches basées sur les algorithmes génétiques, les algorithmes NSGA-II Deb, [41] et SPEA-II Zitzler, [42] sont actuellement considérés comme les plus performantes d'après Coello, [29]. Il s'agit généralement des méthodes les plus utilisées de par leur capacité à bien fonctionner sur un grand nombre de problèmes.

Malgré la quasi-universalité de ces approches, certains auteurs regrettent dernièrement le manque de réelle avancée depuis la publication de ces deux approches : « New algorithms can be designed, but they require fresh ideas rather than small (and little innovative) changes to existing approaches, which is a common pattern in much of the research that we see nowadays » Coello, [43].

Il est cependant très difficile d'adapter ces approches au problème du plus court chemin. Il est en effet assez difficile de modéliser le problème dans le cadre précédent à cause du grand nombre de contraintes inhérentes à ce problème : une solution générée aléatoirement n'a pratiquement aucune chance d'être ne serait-ce que réalisable. Même si certains auteurs ont malgré tout essayé d'utiliser ces approches évolutionnaires, la grande majorité des problèmes de plus court chemin multiobjectifs ont été résolus avec des méthodes exactes.

Une exception notable est Maria, [44], où les auteurs ont utilisé l'algorithme SPEA2 pour le problème du plus court chemin multiobjectif. Les chromosomes sont représentés par des entiers correspondants aux arcs d'un chemin. Les chromosomes sont donc de longueur variable. La population initiale est obtenue en partant de la source, puis en tirant récursivement aléatoirement un nœud parmi les successeurs du dernier nœud jusqu'à atteindre le puits. Le croisement est généré en tirant aléatoirement deux individus jusqu'à ce que les deux individus possèdent un nœud en commun pour servir de pivot au croisement. La mutation est obtenue en sélectionnant aléatoirement un nœud de l'individu et en supprimant tout le chemin à partir de ce nœud jusqu'au puits. Le reste du chemin est reconstruit de la même manière qu'à l'initialisation.

Cette approche a été testée sur des instances aléatoires. Le nombre d'optima dans le front obtenu et la diversité des solutions est étudiée. Les résultats montrent qu'au fur et à mesure des itérations les solutions tendent vers le front de Pareto et que les solutions sont assez diversifiées. On peut cependant regretter que les auteurs n'aient pas testé les résultats sur des instances dont le front de Pareto est connu et que la performance en temps ait été laissée de côté.

Optimisation par essaims particulaires

D'autres approches étudiées récemment telles que les essaims particulaires multiobjectif. Cette métaheuristique, qui a été initialement proposée par Kennedy,Eberhart [45], imite le comportement d'un essaim d'oiseaux capable de trouver une source de nourriture malgré une communication restreinte entre les individus.

Les résultats des variantes multiobjectif sont compétitifs par rapport aux algorithmes plus traditionnels. Une comparaison complète des différentes adaptations possibles a été effectuée dans Durillo, [46].

Cependant il est peu probable que cette approche soit un jour appliquée à un problème de type plus court chemin. En effet, cette approche fonctionne bien sur des problèmes continus avec peu de contraintes sur les variables, tout à l'opposé des problèmes de plus courts chemin qui sont discrets et très fortement contraints.

Optimisation par colonies de fourmis

En s'inspirant de la manière dont les fourmis arrivent à trouver le plus court chemin jusqu'à une source de nourriture, les auteurs de Dorigo, [47] ont réussi à résoudre des problèmes combinatoires tels que le voyageur de commerce.

Il a également été tenté d'adapter l'optimisation par colonies de fourmis pour résoudre des problèmes multiobjectifs. Les différentes possibilités de paramétrage sont étudiées sur le problème du sac à dos multiobjectif dans Alaya, [48]. Cependant les tentatives restent modestes et ne se démarquent pas significativement.

Puisque la métaheuristique s'inspire de fourmis cherchant le plus court chemin, il est naturel de vouloir transposer cette approche à des problèmes de plus court chemin. En effet si le plus court chemin peut être résolu efficacement avec l'algorithme de Dijkstra, ce n'est pas le cas pour le problème du plus court chemin multiobjectif. Cependant, les très rares

tentatives pour résoudre un simple plus court chemin ne permettent que de résoudre des instances tellement petites qu'une adaptation à des situations plus compliquées n'est pas envisageable — en particulier en observant les excellentes performances des algorithmes exacts.

En effet, dans Dorigo,Stützle [49] les auteurs présentent l'approche historique des ACO et montrent comment l'adapter sur un problème de plus court chemin classique. Ils travaillent sur un graphe de vingt nœuds. Il leur faut 1 500 explorations avant de converger vers un résultat et plus de 10 000 avant de trouver l'optimum. Dans Pluciński [50] l'espace — initialement continu — est discrétisé en une grille 20×20, créant ainsi un graphe en forme de grille de 400 nœuds. Des obstacles sont placés dans cet environnement et les arcs correspondants sont supprimés du graphe. Selon les instances, avant de trouver l'optimum, entre 6 000 et plus de 200 000 itérations de 20 fourmis sont nécessaires. Enfin, dans Kolavali,Bhatnagar [51] le but est de résoudre le plus court chemin dans un graphe en couches (4 couches de 3 nœuds, donc avec une source et un puits, 14 nœuds au total). Le nombre d'itérations n'est pas défini « we ran each algorithm for a sufficiently long time » mais avec 32 fourmis l'optimum n'est atteint que entre 55 et 70% de fois (selon les différentes variantes et paramétrages). Avec 128 fourmis les résultats sont meilleurs (entre 89 et 100%).

À titre de comparaison, une implémentation naïve de l'algorithme de Dijkstra explore n^2 nœuds dans le pire des cas. Ainsi avec 400 nœuds, au pire 160 000 nœuds sont visités. Dans le meilleur des cas, 6 000 itérations avec 20 fourmis sont nécessaires. Le colonies de fourmis nécessitent donc plus d'itérations que le nombre de nœuds visités par l'algorithme Dijkstra naïf !

1.4.5.2 Approches exactes

La première approche proposée semble être Hansen [52] pour résoudre le plus court chemin bicritère. L'algorithme de *Martins* en est l'extension immédiate pour un nombre arbitraire d'objectifs.

Algorithme de Martins

Pour le problème du plus court chemin multiobjectif, **l'algorithme 1.4** applé l'algorithme de *Martins* Martins [53] est le plus connu. Il s'agit d'une généralisation de l'algorithme de Dijkstra au cas multiobjectif. Cet algorithme définit une étiquette (*label*) comme un triplet

(nœud, valeurs des objectifs, prédécesseur). À la fin de la résolution, pour chaque nœud il y aura plusieurs labels non dominés correspondants au front de Pareto. Le prédécesseur permet de reconstruire l'itinéraire à suivre pour obtenir une telle valeur d'objectif. Ainsi l'étiquette `(u, c, e)` signifie qu'au nœud u il existe un chemin de coût c en arrivant par l'étiquette e.

Il est nécessaire d'utiliser des étiquettes parce qu'il peut y avoir plusieurs vecteurs de coût acceptables pour chaque nœud. Contrairement à l'algorithme de Dijkstra, il n'est donc pas possible d'associer à chaque nœud un seul prédécesseur et un seul coût. L'ensemble `Q` contient donc un ensemble d'étiquettes. À chaque itération, l'étiquette `(u, c, e)` ayant la plus petite valeur d'objectifs dans l'ordre lexicographique est choisi. Cette étiquette est mémorisé dans une archive `P` car elle correspond à une solution Pareto-optimale. En effet il ne peut pas exister d'autre étiquette ayant un coût lexicographique plus petit ; il s'agit donc d'une solution non-dominée.

L'ordre lexicographique est utilisé pour sa simplicité. Cependant tout ordre *total* aurait pu être utilisé sans changer le résultat obtenu. Cela ne suppose donc pas que certains objectifs soient plus importants que d'autres.

Pour chaque successeur `v`, une nouvelle étiquette est créée `(v, c + c[u][v], e2)`. Enfin, on élimine toutes les étiquettes dominées par une autre.

L'algorithme termine lorsque l'ensemble `Q` est vide. Étant donné que le nombre d'étiquettes d'un nœud peut devenir exponentiel, le temps d'exécution de l'algorithme est donc également exponentiel.

```
1  FONCTION Martins(source, nœuds, arcs, couts)
2      Q := (source, 0, null)
3      P := {}
4      TANT QUE Q est non vide FAIRE
5          e2 = (u, c, e) := étiquette de Q minimisant c
6          Q := Q \ {e2}
7          P := P ∪{e2}
8          POUR TOUT w successeur de v FAIRE
9              Q := Q ∪{(v, c + c[u][v], e2)}
10         ÉLIMINER ÉTIQUETTES DOMINÉS(Q)
```

Algorithme 1.4 Algorithme de Martins

Autres approches exactes

Dans Guerriero,Musmanno [54] les auteurs comparent plusieurs approches de *labelling* pour le plus court chemin multiobjectif. Parmi les méthodes de *labelling*, on distingue deux familles. À chaque itération il est possible de sélectionner un nœud (*node selecting*) ou une étiquette (*label selecting*).

Dans le cadre du *node selection*, l'auteur définit une opération *merge* qui génère le nouvel ensemble Pareto-optimal de tous les successeurs d'un nœud donné. Plusieurs approches sont explorées pour sélectionner le nœud : FIFO, plus petite moyenne des coûts des étiquettes associés à un nœud et une file où les nœuds sont placés en tête si la moyenne des coûts est plus petite et en queue sinon.

Dans le cadre du *label selecting*, choisir la plus petite étiquette dans l'ordre lexicographique correspond à l'algorithme de Martins (*label setting*). Les auteurs introduisent comme moyen de sélection le FIFO et l'utilisation d'une file où une étiquette est mise en tête ou en queue si elle est plus petite que l'étiquette en tête. L'ordre lexicographique et la somme des objectifs sont considérés pour comparer les étiquettes.

Des mesures expérimentales sont faites sur des graphes aléatoires de taille et de densité variables et également sur des graphes en forme de grille. Deux à quatre objectifs sont étudiés. Globalement aucune des approches n'est significativement plus performante que l'autre. On constate tout de même que la technique de *label-setting* se comporte globalement mieux sur les graphes les plus denses et les grilles. Il semblerait que le temps d'exécution soit fortement corrélé avec le nombre d'éléments sur le front de Pareto.

Une approche basée sur la programmation linéaire est utilisée dans Tarapata [55]. Malgré l'utilisation du solveur de programmes linéaires CPLEX réputé pour sa performance, les résultats sont moins intéressants que ceux d'un algorithme de Dijkstra modifié.

Amélioration des performances

Une adaptation de l'algorithme A* est proposée dans Stewart,White III [56]. La fonction heuristique retourne une estimation inférieure pour chaque objectif, permettant ainsi de guider la recherche. Les auteurs présentent plusieurs heuristiques ainsi que les conditions qu'elles doivent remplir. Cette approche est appelée MOA*. Elle se base sur la sélection des nœuds. Dans Mandow,de la Cruz [57] les auteurs présentent une extension de A* qui sélectionne les étiquettes et non pas les nœuds, contrairement à l'approche précédente.

Dans Sauvanet,Néron [58], les auteurs sélectionnent un certain nombre de nœuds pour calculer les plus courts chemins entre toutes les paires afin d'obtenir de meilleures bornes et ainsi réduire le nombre de nœuds explorés. L'application permet de calculer le plus court chemin bicritère (durée et sentiment de sécurité à vélo) à l'échelle d'un département (environ 100 000 nœuds en moins d'une seconde).

Les résultats les plus performants sont ceux de Delling,Wagner [59]. Leurs résultats sont cependant à modérer à cause de la nature des objectifs considérés. Ceux-ci sont en effet très fortement corrélés (distance et temps par exemple) ne générant donc que très peu d'éléments dans le front de Pareto.

Dans Geisberger, [60], les auteurs adaptent les *contraction hierarchies* de Geisberger, [5] pour proposer un calcul d'itinéraire multiobjectif dont la fonction de coût est linéarisée *a priori*. Étant donné que cette approche se base sur un pré-traitement, les coefficients possibles doivent être connus à l'avance. Cela apporte donc une limitation supplémentaire par rapport à la linéarisation des objectifs. L'avantage de cette approche — si le cadre d'application tolère une limitation aussi forte — réside dans des performances encore meilleures que celles de Delling,Wagner [59]. Un calcul à travers l'Europe ne nécessite en effet qu'environ 10 ms.

Modification des paramètres

Il se peut que les paramètres changent de manière imprévue (retards dus à une panne ou à un incident météo). D'après ses auteurs, le premier article à traiter les imprévus dans le cas multiobjectif est Berger, [61]. Ils proposent une approche appelée *SUBITO* pour « SUBstructure In Time-dependent Optimization ». Elle consiste à calculer dans un premier temps l'itinéraire théorique le plus rapide et à fixer un temps maximum qui est accepté pour arriver après le temps théorique. Cela permet d'éliminer plus tôt des étiquettes lors du calcul des itinéraires. Le temps maximum de retard considéré est de 60 minutes pour des trajets allant de 2 à 12 heures.

Cette approche est plutôt simple, robuste aux changements qui ont lieu sur le graphe et les expériences montrent une amélioration par un facteur 10 environ par rapport à l'algorithme de Martins.

1.4.5.3 Dominance relâchée

Dans Muller-Hannemann,Schnee [62] les auteurs utilisent une notion assez souple de la dominance afin d'éliminer des solutions peu acceptables par un être humain. Ils appellent

cela la *relaxed Pareto dominance*. Ainsi une solution mettant une minute de moins, mais coûtant dix euros de plus sera considérée comme dominée. Cette approche nécessite donc l'intervention *a priori* du décideur afin de déterminer les paramètres de la dominance et une intervention *a posteriori* pour choisir la solution. L'avantage de cette solution est la réduction du front de Pareto généré et donc la réduction pu temps d'exécution. Une fonction h permet de déterminer si une solution $\mathbf{u}$ domine $\mathbf{v}$: si pour chaque objectif on a $u_i + h_i(\mathbf{u}, \mathbf{v}) \leq v_i$ pour tout $1 \leq i \leq k$ et que pour au moins un i, $u_i + h_i(\mathbf{u}, \mathbf{v}) < v_i$. Cette définition est très large et permet une définition très souple du coût acceptable pour un objectif en fonction de tous les objectifs.

Cette approche a également l'avantage de ne présenter à l'utilisateur (le décideur *a posteriori*) un nombre réduit de solutions au détriment de la perte d'une solution qui aurait pu être préférée. Cependant, si elle permet de réduire l'exploration en éliminant au plus tôt certaines solutions, il n'est pas possible de quantifier la réduction d'éléments dans le front de Pareto. De ce fait, la complexité reste exponentielle dans le cas général.

1.4.5.4 ϵ-optimalité

Ce concept redéfinit la notion de dominance et peut ainsi être considéré comme un cas particulier de la dominance relâchée vue précédemment. Cependant celle-ci présente des résultats théoriques particulièrement intéressants. Au lieu de considérer une inégalité stricte afin de savoir si une solution domine une autre, une tolérance à un facteur ϵ_i près sur chaque objectif i est acceptée. La dominance est donc formellement définie de la manière suivante :

$$\mathbf{u} \prec_\epsilon \mathbf{v} \Leftrightarrow \forall i \in \{1, \ldots, k\},\ u_i \leq (1 + \epsilon_i) \cdot v_i,\ \exists i \in \{1, \ldots, k\}\ u_i < v_i$$

Par souci de simplicité, nous avons considéré par la suite que le ϵ était le même pour tous les objectifs. Cependant les résultats s'appliquent de la même manière en ayant un ϵ_i différent pour chaque objectif.

Avec cette nouvelle définition de la dominance, on peut définir l'ϵ-front de Pareto $\mathcal{PF}^*_\epsilon$, dont toutes les solutions sont au plus à un facteur $1 + \epsilon$ près du front de Pareto optimal pour chaque objectif.

Un certain nombre de résultats remarquables ont été formalisés récemment. Ainsi dans Papadimitriou,Yannakakis [63] les auteurs démontrent que le nombre d'éléments dans le front de Pareto est un polynôme en $\frac{1}{\epsilon}$ et la taille du problème (mais exponentiellement

en fonction du nombre d'objectifs). Ce résultat peut être compris intuitivement en découpant l'espace des objectifs en hypercubes de largeur ϵ : il suffit alors de trouver une seule solution par hypercube.

Même si le front de Pareto a une taille polynomiale, le temps nécessaire pour le calculer ne l'est pas nécessairement. Les auteurs présentent plusieurs conditions pour l'existence d'un tel algorithme. Le théorème central stipule qu'un tel algorithme existe si et seulement s'il existe un autre algorithme permettant de résoudre en temps polynomial le problème suivant : étant donné un vecteur $\mathbf{b}$, retourner une solution $\mathbf{s}$ telle que $f(\mathbf{s}) \prec_\epsilon \mathbf{b}$ ou prouver qu'elle n'existe pas.

L'idée de la démonstration est d'appliquer cet algorithme à chaque hypercube afin de savoir s'il contient une solution. Comme le nombre d'hypercubes est polynomial et que l'algorithme l'est aussi, cette opération reste en temps polynomial. Il suffit à la fin de ne garder que les solutions non dominées.

Application au problème du plus court chemin

Le plus court chemin multiobjectif a déjà été résolu en temps polynomial à un facteur ϵ près dans Warburton [64]. L'algorithme proposé fonctionne sur des graphes acycliques avec des poids à valeurs entières. La complexité est en $O(n^3 \cdot \left(\frac{n \log(nC_{max})}{\epsilon}\right)^{k-1})$ où k est le nombre d'objectifs et C_{max} est le plus grand poids sur l'ensemble des arcs.

Une nouvelle approche a été proposée dans Tsaggouris,Zaroliagis [65] qui est une adaptation de l'algorithme de Bellman-Ford. Le principe de l'algorithme est particulièrement simple et permet de résoudre un des objectifs de manière optimale. La complexité est légèrement améliorée par rapport à celle de l'algorithme de Warburton : $O(nm \cdot \left(\frac{n \log(nC_{max})}{\epsilon}\right)^{d-1})$. Les auteurs ont étendu cette approche au problème de plus court chemin dont les poids ne sont pas additifs (*Non-Additive Shortest Paths*), tels que des problèmes Min-Max.

Détails de l'algorithme de Tsaggouris

Étant donné un graphe dont l'ensemble des arcs est $\mathcal{A}$ et l'ensemble des nœuds $\mathcal{N}$. On souhaite optimiser d objectifs, le d-ième objectif étant résolu de manière optimale. Un chemin $p = (a_1, \ldots, a_l)$ est représenté par une étiquette qui est un triplet ($\mathbf{c}(p)$, pred(p), dernier arc(p)), où $\mathbf{c}$ est la fonction objectif qui renvoie le vecteur de coût du chemin, pred(p) est l'étiquette prédécesseur et dernierarc(p) est le dernier arc a_l du chemin p. À partir d'une étiquette il est donc possible de reconstruire le chemin en remontant les prédécesseurs.

Un vecteur $\mathbf{r} = [r_1, \ldots, r_{d-1}, 1]$ définit un rapport d'erreur acceptable pour chaque objectif. Pour le dernier objectif, ce rapport vaut 1, puisqu'on désire le résoudre à l'optimum.

On note C_i le rapport entre le plus grand poids du graphe et le plus petit pour l'objectif $i : C_i = \frac{\max_{a \in \mathcal{A}} c_i(e)}{\min_{a \in \mathcal{A}} c_i(e)}$. L'espace des objectifs est alors découpé en hypercubes de manière à ce le rapport entre une solution $\mathbf{x}$ et les bords de l'hypercube soit au plus r_i dans chaque dimension i. De ce fait il y a $\log_{r_i}(nC_i)$ découpages par dimension.

On souhaite avoir pour chaque nœud au plus un chemin (donc une étiquette) par hypercube. À chaque nœud on associe donc une matrice `labels` de dimensions :

$$[0..\lfloor \log_{r_1}(nC_1) \rfloor, \ldots, 0..\lfloor \log_{r_d - 1} \frac{c_{d-1}(p)}{c_{d-1}^{min}} \rfloor]$$

$\lfloor x \rfloor$ désigne la troncature de x.

La position d'une étiquette q dans la matrice est obtenue avec la fonction `pos` qui renvoie un vecteur contenant l'indice de chaque colonne :

$$\text{pos}(q) = [\lfloor \log_{r_1} \frac{c_1(p)}{c_1^{min}} \rfloor, \ldots, \lfloor \log_{r_d - 1} \frac{c_{d-1}(p)}{c_{d-1}^{min}} \rfloor]$$

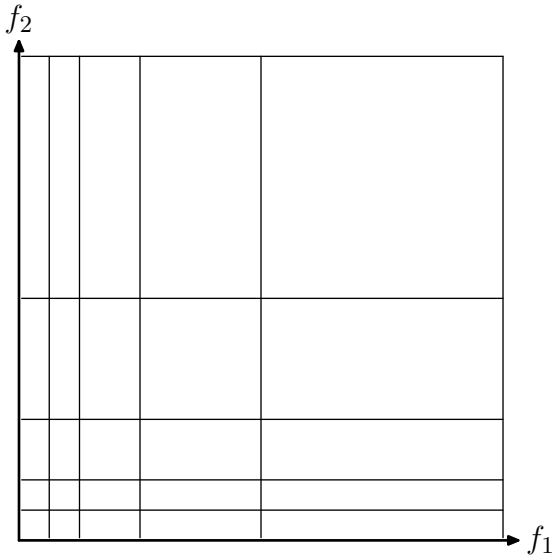


Figure 1.10 Découpage de l'espace des objectifs en ϵ-optimisation. Dans chaque hypercube, nous ne souhaitons garder qu'une seule solution qui sera au plus à un facteur ϵ près des bords de l'hypercube

L'algorithme de Tsaggouris (**algorithme 1.6**) se comporte ensuite comme l'algorithme de Bellman-Ford (**algorithme 1.2**), sauf que les lignes 8 à 10 sont remplacées par la fonction

`fusion` (**algorithme 1.5**) qui compare toutes les étiquettes du nœud précédent afin de voir s'il y a une amélioration.

```
1 FONCTION fusion(R, Q, a)
2     POUR TOUT p de Q FAIRE
3         q := (c(p) + c(a), p, a)
4         si R[pos(q)] = null OU c_d(R[pos(q)]) > c_d(q) ALORS
5             R[pos(q)] := q
6     RETOURNER R
```

Algorithme 1.5 Procédure `fusion` de l'algorithme de Tsaggouris

```
1 FONCTION Tsaggouris(source, nœuds, arcs, poids)
2     POUR TOUT v de nœuds FAIRE
3         label[v] = null
4     label[source] = (0, null, null)
5     RÉPÉTER |nœuds| FOIS
6         POUR TOUT a = (u,v) de arcs FAIRE
7             label[v] := fusion(label[v], label[u], a)
```

Algorithme 1.6 Algorithme de Tsaggouris

1.5 Itinéraires multimodaux

Un itinéraire multimodal est un itinéraire combinant différents modes de transports tels que la marche, le métro ou encore le vélo. Il n'existe pas de réel consensus sur la définition exacte. Ainsi certaines publications préfèrent utiliser le terme *intermodal*. De plus, certains auteurs considèrent que la combinaison de plusieurs lignes de transports en commun est déjà un itinéraire multimodal en soi.

Puisqu'on prend en compte les transports en commun, il s'agit d'un problème de chemin de moindre coût en fonction du temps. La difficulté supplémentaire vient du fait de devoir gérer en plus différents modes de transport. En effet, si pour parcourir un arc on possède deux modes de transport, alors se pose le problème de la contrainte FIFO : vaut-il mieux attendre à un nœud (pour prendre un bus), ou partir au plus tôt (à pied) ?

Comme le soulignent Boussedjra, [66], il s'agit d'un domaine où il existe de nombreux outils de simulation, mais paradoxalement très peu de recherche du point de vue optimi-

sation. Cela est d'autant plus étonnant en considérant le potentiel important que de tels outils pourraient apporter dans le domaine du transport de marchandises ou de passagers et ce aussi bien du point de vue des coûts, de la fiabilité, du confort ou encore de l'impact environnemental.

1.5.1 Difficultés

Les difficultés liées aux itinéraires multimodaux sont définies dans Ziliaskopoulos,Wardell [67] :

- les horaires ne sont pas des fonctions continues ;
- certains nœuds sont l'interconnexion de plusieurs lignes opérant sur des réseaux différents, nécessitant de considérer chaque ligne comme un mode différent ;
- nécessité de prendre en compte le temps d'attente aux nœuds ;
- violation de la contrainte FIFO sur certains arcs.

Un récent article Bast [68] étudie toutes les approches utilisées pour améliorer les performances dans le cadre d'un réseau routier et montre que l'application de celles-ci n'est pas toujours possible sur des transports en commun suivant une grille horaire. Dans tous les cas, les performances obtenues ne sont pas aussi performantes.

Dans le cadre d'un calcul d'itinéraire multimodal, les algorithmes doivent être un compromis entre les approches performantes sur un réseau routier et celles performantes avec une grilles horaires. De ce fait, obtenir un calcul d'itinéraire multimodal est probablement plus difficile que d'obtenir un calcul d'itinéraire avec un mode isolé.

1.5.2 Graphe multivalué

Un graphe multivalué consiste à considérer que pour un arc, il y a autant de coûts que de modes de transport sur cet arc. Cette formulation spécifique au transport multimodal permet une représentation compacte, donc plus économe en mémoire, mais en contre partie nécessite des algorithmes capables de gérer cette structure de données.

Les auteurs montrent dans Ziliaskopoulos,Wardell [67] que l'utilisation du modèle habituel *time-expanded*, avec n nœuds, m arcs, q intervalles de temps, o modes de transport,

le graphe espace-temps généré aura noq nœuds et $mqo + no^2q^2$ arcs. Il en résulte qu'un algorithme utilisant un tel graphe aura une complexité en au moins $O(no^3q^3)$.

Les auteurs proposent un algorithme en $O(q^2n^5)$ capable de calculer le chemin le plus rapide même s'il existe des arcs non-FIFO sans nécessiter de générer le graphe espace-temps. Les auteurs considèrent que le réseau piéton est agrégé en quelques points d'intérêt réduisant ainsi significativement le nombre de nœuds du réseau. La conséquence de cela est la multiplication d'arcs non-FIFO, puisqu'il sera probable que l'arc entre deux stations de transports en commun soit utilisable par les piétons. Malgré le facteur n^5, les auteurs montrent expérimentalement que le temps d'exécution augmente de manière pratiquement linéaire en fonction du nombre de nœuds. De plus le temps d'exécution est effectivement insensible au nombre de modes considérés. Cependant les instances sont relativement petites, puisque le nombre de nœuds pris en compte varie entre 25 et 1 000 tandis que le nombre d'intervalles de temps considérés va de 50 à 250. Le temps d'exécution dépasse souvent la dizaine de secondes.

L'approche utilisée dans Boussedjra, [66] est assez similaire. Plutôt que de vouloir minimiser le temps de parcours, le but est de minimiser le nombre de transbordements. La complexité obtenue est en $O(n^2o^2q^2)$. L'algorithme a été testé sur des instances générées aléatoirement. Pour des instances à 100 intervalles de temps et entre 50 et 1 000 nœuds, le temps d'exécution est de l'ordre de la seconde. On peut cependant regretter le manque de réalisme des instances, puisque le nombre minimal de transbordements oscille entre 16 (pour 50 nœuds) et 102 pour 1 000 nœuds.

1.5.3 Approche time-expanded

En se basant sur l'algorithme *Chrono-SPT*, dans Lozano,Storchi [69] les auteurs s'intéressent au plus court chemin *acceptable*. En effet, un itinéraire comportant cinq changements ne sera pas considéré comme acceptable, même si le temps de parcours est plus faible. Il s'agit dans une certaine mesure d'une approche multiobjectif, même si elle est relativement limitée. En effet, seul le coût et le nombre de transferts sont considérés. Il n'est pas possible d'associer plusieurs coûts à un arc.

Le chemin multimodal le plus rapide sur le réseau routier, ferroviaire et aérien est calculé dans Delling, [70] avec d'excellentes performances (quelques millisecondes à l'échelle de l'Europe). Afin de modéliser les transitions acceptables (par exemple il n'est pas possible de prendre une voiture entre deux transports en commun), ils se basent sur une approche proposée dans Barrett, [71]. Chaque arc possède une étiquette (voiture, train, avion, etc.)

et l'algorithme proposé calcule le plus court chemin de manière à ce qu'il soit validé par un automate fourni en paramètre (par exemple (marche ou voiture) puis avion puis marche) pour un trajet qui commence par de la marche ou en voiture, continue en avion et finit par de la marche). Les bonnes performances sont à relativiser puisque la part de transports en commun est particulièrement faible. Ansi dans leur plus grande instance, qui couvre l'Europe et l'Amérique du nord, les auteurs ne prennent en compte que 359 aéroports. À titre de comparaison, la région Île-de-France comporte plus de 25 000 arrêts de bus. Le fait que les transports en commun soient si peu représentés permet aux auteurs d'utiliser des algorithmes de plus courts chemins traditionnels pour obtenir d'aussi bonnes performances.

1.5.4 Approche multiobjectif

Dans Meng, [72], le plus court chemin multimodal multiobjectif est calculé dans le réseau de transports en commun de Singapour (environ 3 000 nœuds). Cependant seul les réseaux de bus et de métro sont pris en compte. Des arcs de marche sont rajoutés pour connecter les stations proches. L'aspect multiobjectif est traité en linéarisant le temps de parcours et le coût. La résolution se fait avec une version modifiée de l'algorithme de Dijkstra.

Dans Boussedjra, [73], les auteurs linéarisent les objectifs et utilisent un algorithme génétique pour faire varier les poids dans le but de construire le front de Pareto. Les individus sont donc représentés par les poids de la fonction objectif. L'évaluation d'un individu consiste à calculer le plus court chemin mono-objectif avec les poids associés. L'article se focalise sur l'étude des possibilités de l'opérateur de diversification, afin d'éviter de converger dans la même zone du front de Pareto. Les performances sont assez modestes. Si l'opérateur proposé (basé sur une liste tabou) permet d'augmenter la qualité des solutions trouvées, le réseau étudié est plutôt petit : 60 nœuds, 9 à 24 modes par nœud et 200 dates de départ par mode et nœud.

Une première approche réellement multiobjectif a été proposé dans Gueye, [74], Gueye, [75]. L'approche proposée modélise les transitions possibles entre modes grâce à un nombre réduits d'états dans lequel se trouve l'utilisateur en fonction des modes qu'il a déjà emprunté et du mode en cours. Les algorithmes proposés ne fonctionnent que si le deuxième objectif est une variable entière prenant un nombre réduit de valeurs (par exemple le nombre de changements). Pour une ville de la taille de Toulouse, le temps de calcul est de l'ordre de la seconde. Cependant il semblerait que le temps de parcours en

transports en communs ne soit qu'une approximation du temps de parcours et ne prenne pas réellement en compte les grilles horaires.

1.5.5 Questionnement

On peut s'étonner du peu de recherche faite dans ce domaine. En effet, les résultats actuels, même récents sont particulièrement médiocres ou peu adaptés pour une mise en application sur des réseaux de grande taille tel que celui de la région Parisienne.

Les approches multimodales les plus performantes se contentent de calculer le chemin le plus rapide et ne permettent pas de prendre en compte d'autres coûts. De plus elles ne s'intéressent pas aux réseaux avec une forte interaction entre transports en commun ni aux modes utilisant le réseau routier.

En ce qui concerne les approches multiobjectif, elle ne prennent en compte que très rarement l'aspect multimodal et ont des temps de calcul peu satisfaisants. À notre connaissance, personne n'a réellement abordé les problèmes liés à la dépendance du temps.

En particulier, il est surprenant que les deux approches habituellement utilisées dans les graphes à grille horaire, à savoir *time-dependent* et *time-expanded* n'aient pas été utilisées pour le chemin multiobjectif. En effet, l'utilisation de graphes multivalués nécessite l'utilisation d'algorithmes qui se révèlent particulièrement compliqués.

Conclusion

La recherche bibliographique a montré qu'il existe de nombreux algorithmes de calcul de plus court chemin, mais il reste encore une certaine marge de progression en ce qui concerne la modélisation d'un réseau réellement multimodal et de l'utilisation des techniques d'optimisation multiobjectif.

Le chapitre suivant fixe la problématique de cette thèse à la lumière des publications les plus proches de nos travaux.

Chapitre 2
Problématique

Neo, sooner or later you're going to realize just as I did that there's a difference between knowing the path and walking the path.

— Morpheus in Matrix

Introduction

Ce chapitre essaye de présenter dans un premier temps les spécificités du problème que nous essayons de résoudre, à savoir le plus court chemin multiobjectif dépendant du temps. Dans un second temps, nous présenterons dans quelles situations les algorithmes peuvent être appliqués. Enfin, nous tenterons de qualifier expérimentalement le nombre de solutions Pareto-optimales trouvées en fonction de la nature des fonctions objectifs lors d'un calcul du plus court chemin.

2.1 Problématique

2.1.1 Intérêt pour l'utilisateur

Les calculateurs d'itinéraires multimodaux actuellement disponibles sont particulièrement simplistes. En effet ils ne proposent à l'utilisateur qu'une seule solution. Lorsque plusieurs solutions sont présentées, il ne s'agit que de solutions décalées dans le temps et ne forment pas nécessairement un ensemble Pareto-optimal (par exemple sur le site de vente en ligne de la SNCF).

Les contraintes peuvent être parfois définies par l'utilisateur mais restent très restreintes. Ainsi, il est généralement possible de choisir uniquement les modes que l'on souhaite prendre. Quant aux objectifs à optimiser, ils sont toujours limités au temps ou au nombre de changements.

Nous pensons qu'il est préférable d'avoir un système ayant le moins de paramétrage possible (idéalement aucun), tout en convenant à l'utilisateur dans toutes les circonstances. Il n'est donc pas possible de préjuger des préférences de l'utilisateur et il est indispensable de lui proposer plusieurs solutions alternatives (par exemple est-ce que cela gênerait l'utilisateur de marcher une dizaine de minutes ?).

La nécessité de présenter plusieurs solutions pour un même utilisateur est d'autant plus vraie qu'une même personne ne prendra pas nécessairement d'un jour à l'autre le même itinéraire pour un même trajet. En effet, selon la météo, la charge, le temps disponible, une personne choisira le vélo, la voiture, le métro — ou une combinaison de ces modes. Ces données étant impossibles à modéliser, il est nécessaire de laisser à l'utilisateur le choix. De plus cette approche permettra à l'utilisateur d'améliorer son itinéraire en découvrant une possibilité qu'il ne voyait pas auparavant.

2.1.2 Intérêt scientifique

Notre sujet se rapproche de nombreuses approches présentées au premier chapitre. Cependant, nous essayons d'opter pour une approche globale prenant en compte simultanément la multimodalité et l'optimisation multiobjectif. En effet les calculs d'itinéraires multimodaux ne sont que très rarement multiobjectifs et les seules approches multiobjectif n'ont été faites que sur des régions particulièrement petites.

Hétérogénéité des modes de transport

Les approches proposées ne considèrent généralement pas les modes de transport hétérogènes tels que les vélos en libre service. De plus elles se contentent de prendre en compte les transports en commun et, dans une mesure très limitée, la marche. L'introduction de nouveaux modes de transport multiplie le nombre d'interconnexions et donc le nombre d'itinéraires *équivalents* possibles. Le nombre de modes accroît donc significativement la complexité du problème, tout particulièrement avec une approche multiobjectif.

Pas d'approximations

Certaines approches agrègent certaines zones en un seul nœud rendant ainsi les résultats approximatifs. De plus l'approche *time-expanded* qui est souvent utilisée pour les transports en commun n'est pas valable pour le transport routier sans introduire une forte

approximation du temps de parcours sur les arcs ou de considérer un nombre considérable d'intervalles de temps (chaque intervalle devrait durer de l'ordre de la seconde).

Des fonctions de coût difficiles

En ce qui concerne les approches multiobjectif, la nature des objectifs considérés ne génère généralement que peu de solutions Pareto-optimales. En effet, plus il y a de changements, plus il y a de chance que le trajet soit long. De même le temps parcouru est très fortement corrélé à la distance parcourue. *A contrario*, les émissions de CO_2, le dénivelé positif ou encore le coût monétaire génèrent considérablement plus de possibilités de parcours si l'on prend en compte des modes de transport aussi différents que la marche, le vélo, la voiture et les transports en commun et que l'on s'autorise toute combinaison de ces modes.

Simplicité de l'approche

Toutes les approches proposent des algorithmes assez complexes et donc difficiles à implémenter. Dans le contexte de thèse cofinancée par un industriel, une application réelle devait être aisée. Afin de permettre une évolution de l'application, l'algorithme proposé doit être suffisamment simple à comprendre pour une prise en main rapide et suffisamment générique pour s'adapter à l'évolution des besoins.

Choix des solutions à présenter

Finalement, il sera important de réfléchir à la manière dont seront présentés les résultats à l'utilisateur final. Il est probablement peu souhaitable de lui présenter le front de Pareto *in extenso*, mais il serait également dommage de ne lui présenter qu'une seule solution. De ce fait il sera nécessaire de définir des métriques adaptées qui permettront de sélectionner quelques solutions intéressantes.

2.1.3 Taille des problèmes

Taille du graphe

Nous nous intéressons tout particulièrement aux trajets multimodaux en milieu urbain. L'échelle des problèmes est donc de l'ordre d'une agglomération. La région Île-de-France

peut-être considérée comme une borne supérieure de la part de sa taille (11 millions d'habitants) et la densité de ses modes de transport. Il existe en effet plus de 25 000 arrêts de bus (pour près de 1 500 lignes) et 1 000 stations de transport en commun ferrés (50 lignes). En ce qui concerne les vélos en libre service, 1 500 stations du vélo *Vélib'* sont en service. Enfin, le réseau routier, il comporte environ 200 000 nœuds et 300 000 arcs.

Objectifs considérés

Contrairement aux approches existantes qui ne traient habituellement que deux objectifs, nous nous proposons de prendre en compte :

- le coût,
- le nombre de changements,
- le confort (les utilisateurs préfèrent généralement les transports en commun sur fer tels que le métro ou le tramway par rapport au bus),
- la pollution (un indicateur très simpliste est la quantité de CO_2 émise),
- l'effort physique (pour un vélo ce serait le dénivelé positif).

Temps de calcul souhaité

Pour un utilisateur qui désire choisir un itinéraire sur un site web, l'attente doit être de l'ordre de la seconde. De ce fait le temps de calcul doit également être de cet ordre de grandeur. Nous nous penchons également sur le cas d'un serveur web devant traiter plusieurs requêtes simultanément.

2.1.4 Comparatif des approches existantes

Nous proposons de reprendre les travaux présentés au premier chapitre afin de les comparer et constater leurs limites respectives. Le **tableau 2.1** indique les points faibles ou non traités (-) et les points forts (+) de chaque publication. Un 0 indique que l'approche proposée traite le problème sans se démarquer.

Nous comparons les éléments suivants :

- Modélisation du problème à résoudre :
  - Transports hétérogènes : est-ce que tout type de transport est modélisable ?
  - Densité multimodale : est-ce que les interconnexions entre modes de transport correspondent à la réalité ?
  - Multiobjectif : plusieurs objectifs sont optimisés simultanément ?
- Algorithmes proposés :
  - Taille des instances : est-ce qu'elles correspondent à celle d'une grande ville ?
  - Approche exacte : est-il nécessaire de faire des approximations ?
  - Simplicité : est-ce que les algorithmes sont simples à prendre en main ?

Référence	Hétérogène	Densité	Multiobjectif	Taille	Exact	Simple
Pallottino,Scutella [20]	+	+	-	-	-	-
Chabini [27]	+	+	-	-	-	-
Meng, [72]	0	+	-	0	+	+
Ziliaskopoulos,Wardell [67]	+	+	-	-	-	-
Lozano,Storchi [69]	+	+	+	-	-	-
Boussedjra, [66]	+	+	-	-	-	-
Boussedjra, [73]	+	+	+	-	-	-
Muller-Hannemann,Schnee [62]	-	-	+	+	+	+
Delling, [70]	+	-	-	+	+	-
Bast, [21]	-	+	+	+	0	-

Tableau 2.1 Comparaison des publications existantes, points forts(+) et points faibles(-)

On constate qu'aucune approche ne couvre réellement tous les objectifs que nous nous sommes fixés. Notre approche vise à être suffisamment générique et performantes afin de remplir ces six critères.

2.2 Limite d'application des algorithmes

2.2.1 Plus court chemin statique

L'algorithme de Dijkstra n'est valable tant que les coûts sont positifs. L'algorithme de Bellman-Ford permet de détecter des cycles absorbants. Cependant en cas d'existence de tels cycles, l'algorithme n'est pas applicable puisque trouver le plus court chemin élémentaire dans ce cas est un problème difficile.

2.2.2 Chemin le plus rapide en fonction du temps

Nous avons vu que l'algorithme de Dijkstra est valable tant que la fonction de coût est croissante et tend vers l'infini. Lorsque le coût représente le temps, les fonctions ont toujours la forme

$$f^t(t) = t + p(t)$$

où $p(t)$ est le temps de parcours de l'arc. Puisque ce temps de parcours est positif, la fonction est strictement croissante et tend vers l'infini.

Il est donc possible d'utiliser l'algorithme de Dijkstra dépendant du temps à condition que le temps de parcours d'un arc soit strictement positif.

2.2.3 Chemin de moindre coût dépendant du temps

Lorsque le coût dépend du temps, les conditions sur les fonctions de coût sont plus restrictives. En effet, en plus de la complexité théorique (problème NP-difficile), la solution peut être un chemin de longueur infinie, même si tous les coûts sont strictement positifs. La **figure 2.1** est un exemple directement inspiré de Orda,Rom [25] où les auteurs donnent un exemple simple où le chemin de moindre coût est de longueur infinie.

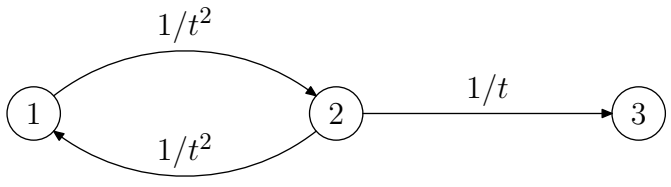


Figure 2.1 Chemin de moindre coût de longueur infinie. À chaque parcours d'un arc, t est incrémenté de 1. Le chemin de moindre coût bouclera une infinité de fois entre 1 et 2

Le temps pour parcourir un arc est toujours de 1. Les coûts en fonction du temps sont indiqués sur chaque arc. Le chemin le plus court du nœud 1 au nœud 3 est celui qui boucle infiniment entre les nœuds 1 et 2 avant d'emprunter l'arc (2,3). Le coût optimal est donc de :

$$\sum_{t=1}^{t=+\infty} \frac{1}{t^2} = \frac{\pi}{6}$$

Les auteurs donnent des conditions suffisantes pour que le chemin de moindre coût soit fini :

- le coût de chaque arc a une borne inférieure strictement positive à partir d'un instant T ;
- le coût d'attente à un nœud tend vers $+\infty$ avec le temps.

2.2.4 Plus court chemin multiobjectif dynamique

À notre connaissance, le plus court chemin multiobjectif dont les coûts sont fonction du temps n'a jamais été étudié. Alors que dans le cas précédent (chemin de moindre coût dépendant du temps) il suffisait que les fonctions de coût aient une borne inférieur strictement positive, dans le cas multiobjectif il faut poser des conditions plus strictes. En effet si la fonction est à un moment décroissante et continue, alors il existera un nombre infini d'éléments dans le front de Pareto.

Considérons un péage urbain : pour rentrer dans le centre-ville, sur certains arcs il faut payer un péage. Ce coût peut être fonction de l'heure de la journée pour décourager l'utilisation de la voiture en heure de pointe, comme dans **figure 2.2**.

Dans le cas d'un segment décroissant, attendre un tout petit instant fait économiser un peu d'argent. De ce fait, comme la fonction est continue, il existe une infinité de solutions Pareto-optimales.

Une condition nécessaire pour que le nombre de solutions Pareto-optimales soit fini est que les fonctions de coût soient croissantes en un nombre fini de morceaux, c'est-à-dire qu'en aucun point elle ne peut être décroissante et continue.

2.2.5 Conditions générales

Nous avons vu que les fonctions de coût doivent avoir une borne inférieure pour que le chemin le moins cher soit fini. Dans le cadre multiobjectif, les fonctions de coût doivent

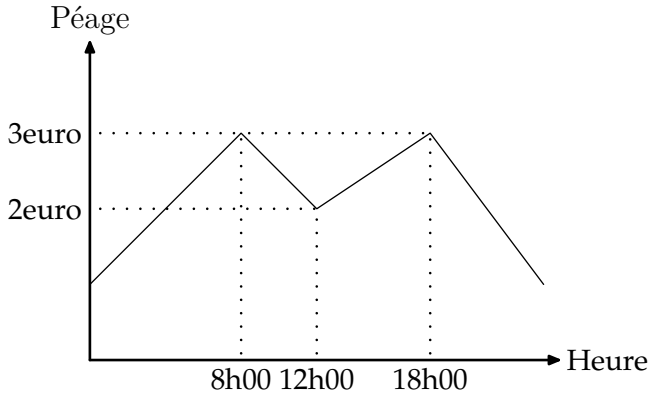


Figure 2.2 Exemple de péage urbain générant un front de Pareto continu. Entre 8h et 12h, attendre un instant de plus réduira le coût du péage. Il y a donc une infinité de solutions Pareto optimales

en plus être croissantes par morceaux pour avoir un nombre fini d'éléments dans le front de Pareto.

Cependant, la majorité des coûts considérés sont constants (en dehors du temps de parcours). En pratique les fonctions de coût remplissent donc les conditions pour qu'un algorithme puisse finir en un temps fini.

2.3 Front de Pareto généré selon les fonctions de coût

Nous souhaitons pouvoir estimer la taille du front de Pareto selon la nature des fonctions de coût et du nombre d'objectifs.

Nous observons également le temps de calcul nécessaire en fonction de la taille du front.

2.3.1 Les graphes

Nous considérons trois graphes : un graphe en forme de grille parfaite où tous les arcs sont de même longueur, un graphe basé sur le réseau routier réel de San Francisco et un autre basé sur le réseau routier de Rennes.

Les graphes comportent respectivement 1 225, 1 259 et 1 239 nœuds. Nous n'avons en effet considéré qu'une petite partie de la ville afin d'avoir des tailles de graphe similaires.

2.3.2 Les coûts

Nous ne considérons que des poids constants. Nous nous intéressons à trois types de coût :

- réels : la valeur est tirée aléatoirement dans l'intervalle $[(1-\epsilon)\cdot l; (1+\epsilon)\cdot l]$ où l est la longueur de l'arc et ϵ un facteur variable entre 0 et 100% ;
- entiers : la valeur est tirée aléatoirement dans l'intervalle $[0; n]$ où n est variable entre 0 et 100 ;
- binaires : la probabilité que le coût soit 1 va de 0 à 100%.

2.3.3 Résultats

Pour chaque valeur, nous avons effectué 10 calculs d'itinéraire en sélectionnant le point de départ aléatoirement. Nous présentons la valeur moyenne.

Nature du graphe

Contrairement à l'idée intuitive, un réseau routier ne peut pas être facilement comparé à un graphe quadrillé. En effet la **figure 2.3** et la **figure 2.4** montrent que le comportement des deux types de graphes n'évolue pas de la même manière en changeant le type de fonction de coûts. En effet dans un graphe quadrillé des fonctions binaires de coût génèrent bien plus de solutions sur le front de Pareto que des fonctions réelles.

Le comportement des algorithmes sur ces deux villes considérées est très proche malgré une topologie assez différente.

Cependant la comparaison est biaisée par le fait que les longueurs sur le réseau quadrillé sont unitaires alors qu'elles sont typiquement de l'ordre de quelques dizaines de mètres sur le réseau routier réel.

Nature des fonctions de coût

Dans le réseau routier, pour des objectifs binaires avec une faible probabilité d'être à 1, alors il existe significativement moins d'éléments sur le front de Pareto (par exemple un objectif qui compte le nombre de changements). De même, lorsque les objectifs sont fortement corrélés (coût réel avec faible variation autour de la longueur), alors il y aura moins d'éléments sur le front (par exemple distance et temps de parcours).

Corrélation entre taille du front de Pareto et temps de calcul

La **figure 2.5** montre le temps de calcul en fonction de la taille du Front de Pareto dans le cas de coûts continus. On constate une forte corrélation entre le temps de calcul et le

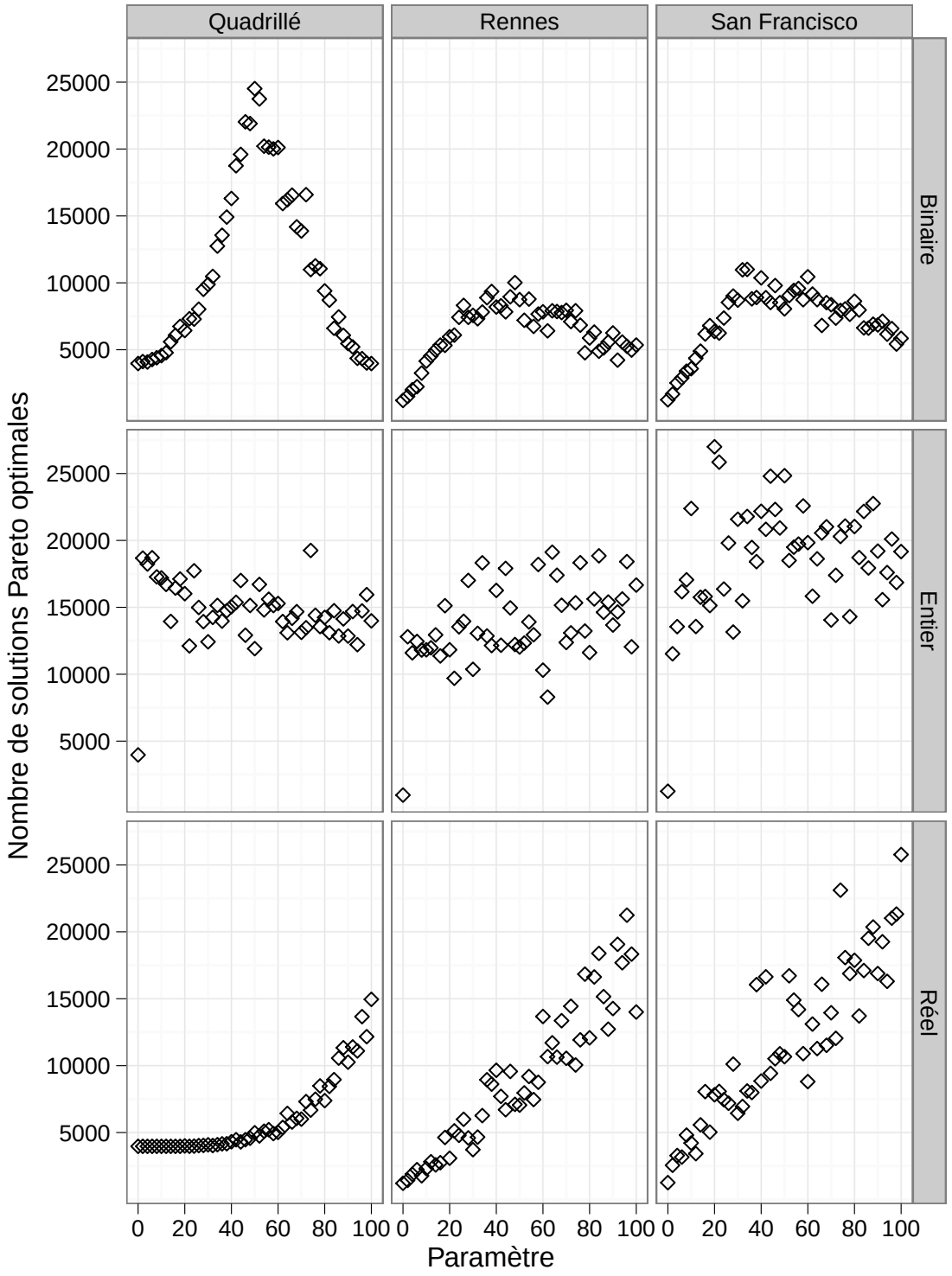


Figure 2.3 Nombre d'éléments dans le front de Pareto en fonction du type de coût et de graphe (2 objectifs)

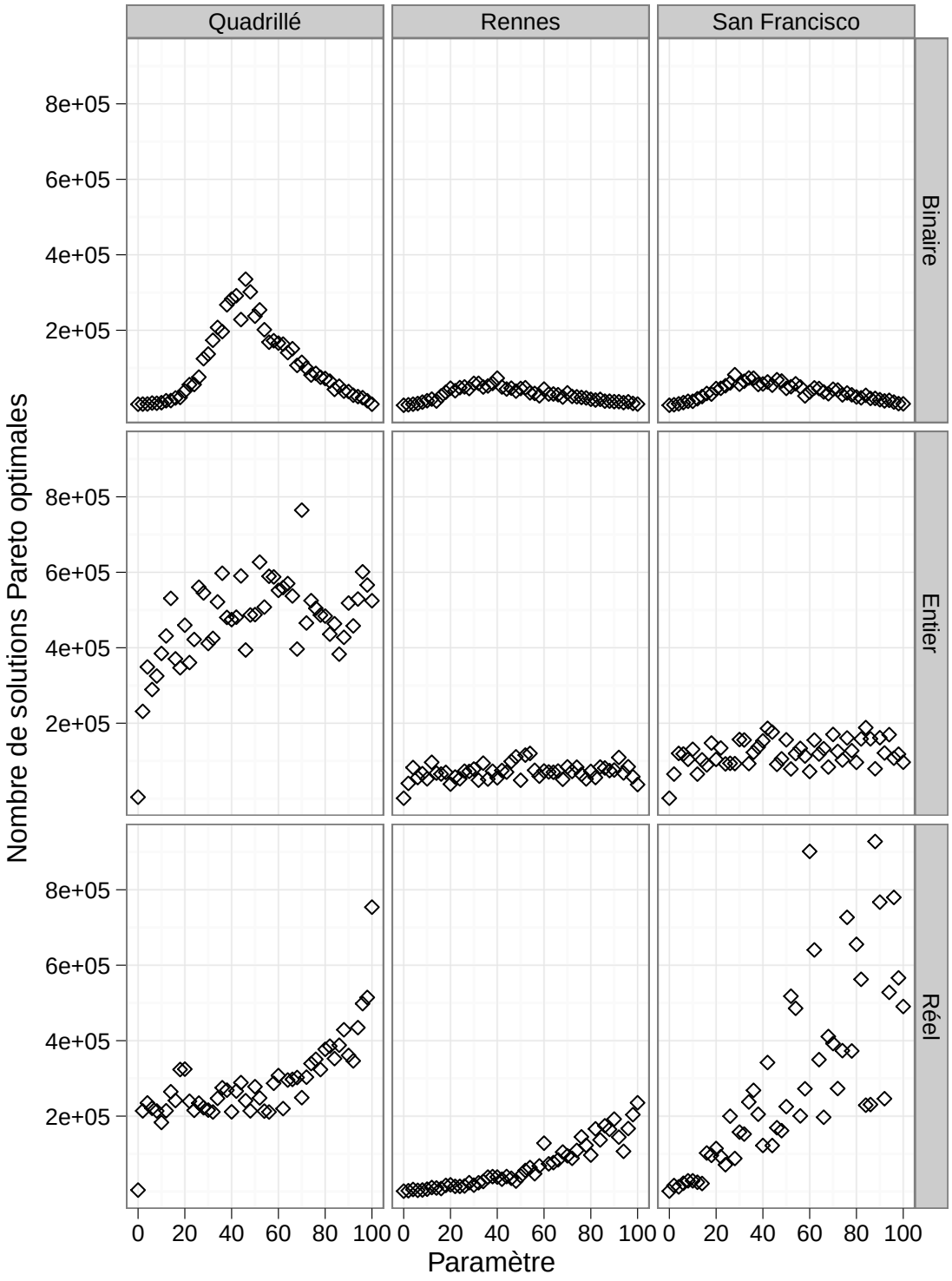


Figure 2.4 Nombre d'éléments dans le front de Pareto en fonction du type de coût et de graphe (3 objectifs)

nombre d'éléments sur le front de Pareto. Afin d'estimer la difficulté d'un problème, la taille du graphe n'est donc pas nécessairement le seul argument pertinent. Il est important de s'intéresser au nombre de chemins équivalents trouvés.

Influence du nombre d'objectifs

Augmenter le nombre d'objectifs rend le problème nettement plus difficile. Plus étonnant, alors que la taille du front de Pareto est multiplié par 20, le temps de calcul est multiplié par 1 000. Cela peut être expliqué d'une part parce que plus d'étiquettes non dominées sont générées pendant la résolution. D'autre part, un profilage montre que l'algorithme passe plus de 90% du temps à tester la dominance entre deux étiquettes.

2.3.4 Conclusion sur la nature des fonctions de coût

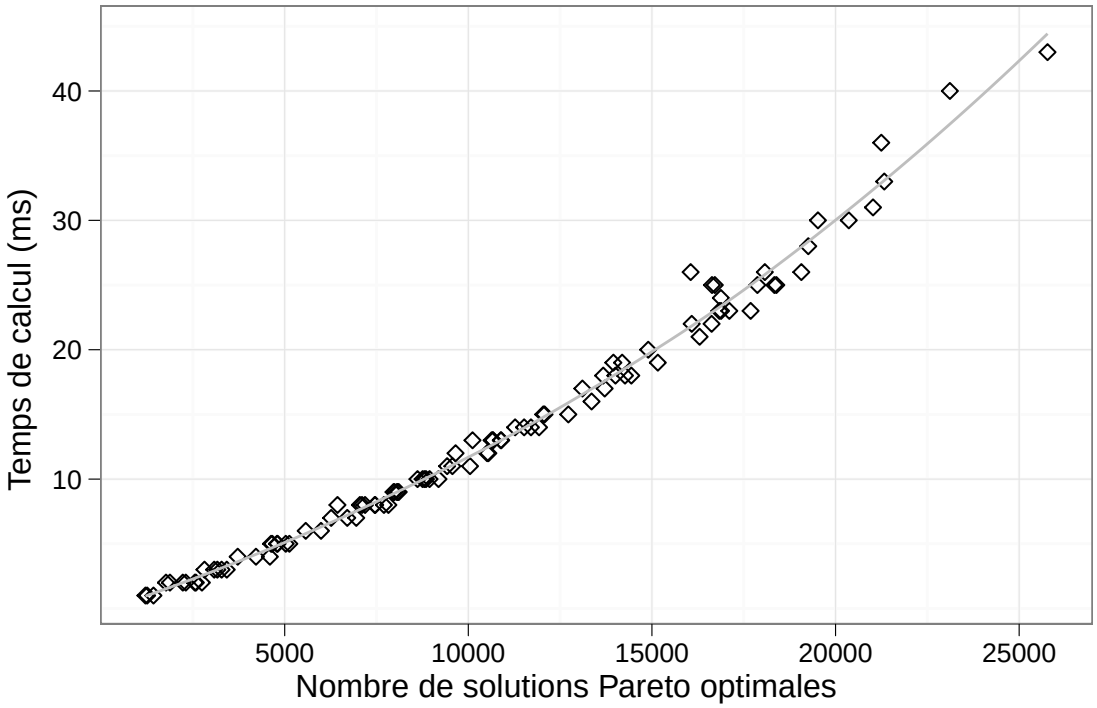
Considérer uniquement le nombre d'objectifs n'est pas pertinent pour avoir une idée de la difficulté du problème. Il faut s'intéresser au nombre d'éléments sur le front de Pareto pour avoir une estimation du temps de calcul. Ainsi, les objectifs généralement considérés (longueur, temps et nombre de changements) génèrent peu de solutions. Il s'agit donc d'un cas facile par rapport à d'autres objectifs (dénivelé, émissions de CO_2). En effet dans le cadre d'un itinéraire multimodal, ces coûts changent drastiquement selon le mode de transport.

2.4 Sélection de certaines solutions

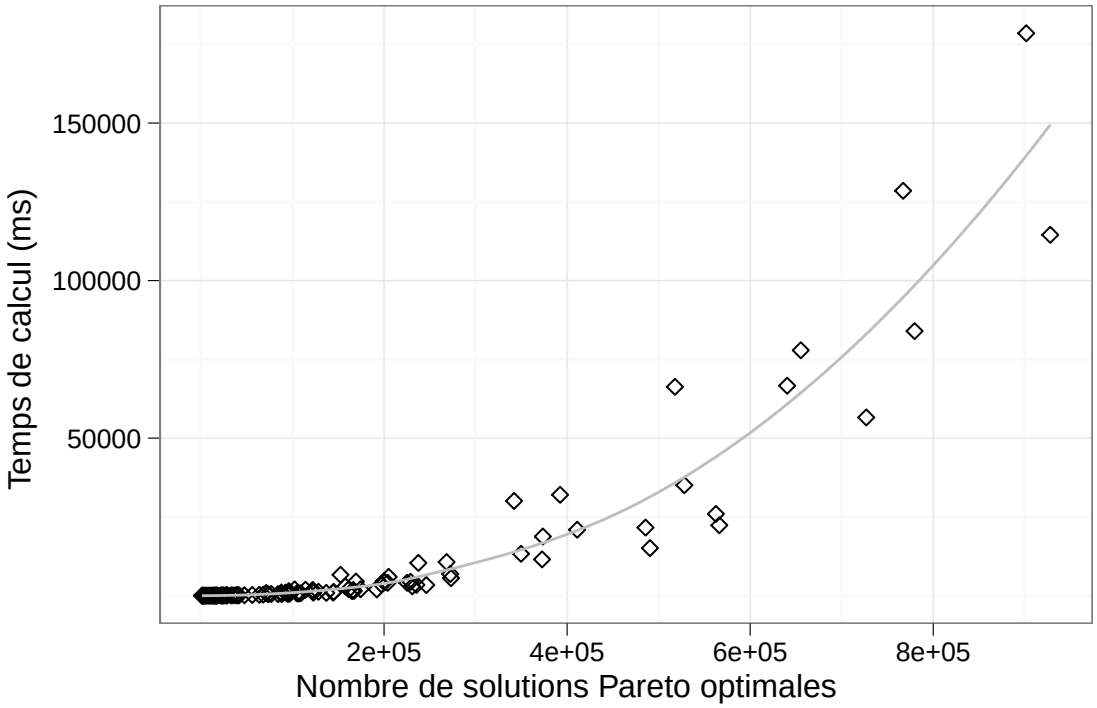
Nous avons vu dans la section précédente que le nombre d'éléments sur le front de Pareto peut devenir particulièrement important (plus de 1 000 solutions par paire de nœuds dans les instances les plus difficiles).

Il est probablement peu pertinent de présenter autant de solutions à un utilisateur final. Il est donc nécessaire de réfléchir sur la manière de ne présenter qu'un nombre réduit de solutions.

Ces solutions présentées doivent être suffisamment diversifiées afin de conserver l'intérêt de l'optimisation multiobjectif. Cette problématique fait l'objet de l'étude dans le chapitre 6.



Deux objectifs



Trois objectifs

Figure 2.5 Temps de calcul en fonction de la taille du front de Pareto

Conclusion

Nous avons présenté dans ce chapitre le problème que nous voulons traiter en comparaison avec les publications existantes. À partir de là, nous avons montré les conditions théoriques que doivent remplir les conditions de coût pour qu'un algorithme trouve un nombre fini de solutions. Cependant, en pratique la large majorité des coûts que nous souhaitons considérer sont constants ou strictement croissants et ne présentent donc pas de problème.

Enfin, nous montrons expérimentalement que la nature des coûts influence considérablement la difficulté du problème. La très large majorité des publications ne s'intéresse qu'à des fonctions de coût faciles (par exemple le nombre de changements), tandis que nous souhaitons intégrer des fonctions de coût plus difficiles (par exemple le dénivelé positif ou les émissions de CO_2.

Maintenant que nous avons défini les limites des publications existantes et les conditions que doivent remplir les fonctions de coût, nous proposons dans le chapitre suivant un modèle générique pour modéliser un réseau multimodal.

Deuxième partie :

Modèle et expérimentations

Chapitre 3
Modélisation d'un réseau multimodal

Simplicity is prerequisite for reliability

— Edsger Wybe Dijkstra

Introduction

Le but de ce chapitre est de proposer une modélisation d'un réseau multimodal qui sera utilisée par la suite dans nos algorithmes. Dans un premier temps seront présentées les caractéristiques souhaitées qui ont abouti au modèle proposé. La deuxième section décrit tous les modes de transport que nous avons envisagés et comment les modéliser. Enfin, des considérations pratiques pour la mise en application seront abordées.

3.1 Caractéristiques souhaitées

3.1.1 Modélisation naturelle

La modélisation seule, d'un réseau routier ou d'un réseau de transports en commun, est quelque chose de très naturel. Cependant, la dépendance au temps complique déjà la situation. En effet des approches telles que l'expansion espace-temps vue dans le **paragraphe 1.3.3.2 page 23** (qui dédouble les nœuds pour chaque intervalle de temps) rendent difficiles la correspondance entre le réseau initial et le graphe généré. La multimodalité pose aussi la question de comment modéliser le fait qu'un même tronçon pourra être emprunté par un bus, une voiture, un vélo ou encore un piéton.

La modélisation doit également être aussi proche que possible de la représentation que l'on se fait naturellement quand il faut se représenter un graphe correspondant à un réseau routier ou à un réseau de bus. Le but est en effet de pouvoir facilement transposer des algorithmes traditionnels pour une implémentation plus simple et une prise en main

plus rapide dans un projet industriel. De ce fait nous ne désirons pas modéliser le temps d'attente à un nœud comme cela a été fait dans d'autres modèles.

3.1.2 Aucune perte d'information

En voulant modéliser tous les déplacements possibles, il est évidemment extrêmement complexe de prendre en compte tous les paramètres. Ainsi, pour un piéton il faudrait être capable de prendre en compte le coté de la route afin de pouvoir savoir le temps nécessaire pour traverser un croisement complexe[10].

Cependant, il serait dommage d'éliminer dès le début (c'est-à-dire à la modélisation) certaines données dans le but de réduire la taille de la représentation, ou d'améliorer les temps de calcul. Ainsi le temps de changement entre deux lignes de métro ou encore le temps nécessaire pour récupérer un vélo en libre service n'est pas négligeable et doit pouvoir être modélisé. Cela interdit donc toute agrégation de nœuds en un seul nœud pour réduire la taille du graphe et cela nécessite de séparer chaque mode de transport pour permettre de prendre en compte le passage d'un mode à un autre.

3.1.3 Prise en compte de tous les modes de transport

Alors que de nombreuses approches se contentent des transports en commun et de la marche, le modèle devra prendre en compte tous les modes de transports individuels (marche, voiture, vélo personnel...), mais aussi les vélo en libre service ou encore les taxis.

3.1.4 Respect de la contrainte FIFO

Utiliser un graphe qui n'est pas FIFO amène des complications significatives du point de vue des algorithmes employés. Afin de ne pas créer de contraintes supplémentaires, le modèle devra être un graphe respectant la contrainte FIFO.

Cela exclut donc la possibilité d'utiliser un graphe multi-valué. En effet si un arc peut être emprunté par un piéton ou par un autobus, il est évident que des situations non-FIFO arriveront.

[10] Cependant des premières applications de guidage pour piétons commencent à apparaître, mais le manque de données risque de les cantonner à zones très réduites

3.2 Modélisation des différents modes

Les souhaits exprimés dans la section précédente amènent naturellement à modéliser le réseau multimodal sous la forme d'un graphe où chaque mode de transport est représenté indépendamment de tout autre mode. Concrètement cela revient à créer un graphe avec plusieurs couches où chaque couche correspond à un mode de transport.

Afin de modéliser le changement d'un mode à un autre, deux nœuds de deux couches différentes peuvent être reliés par un arc lorsqu'un tel changement est possible.

3.2.1 Voitures, piétons et cyclistes

Tous ces modes de transport utilisent le réseau routier et ont une grande liberté de mouvement puisque chaque arc peut être emprunté à n'importe quel instant.

La modélisation du réseau routier est très naturelle. Les nœuds où il y aura des échanges entre les modes dépendront de ce que l'on souhaite modéliser. En considérant qu'une voiture ne peut être laissée que dans un nombre restreint de parkings (par exemples les parkings-relai), les arcs du réseau routier vers le réseau piéton n'existeront que entre les nœuds correspondant à ces parkings. Par contre si on accepte qu'un vélo soit attaché à tout endroit de la ville, il existera un arc depuis chaque nœud de la couche vélo vers la couche piéton.

En ce qui concerne la voiture et le vélo, tout abandon du véhicule est définitif. Il est évidemment impossible de le reprendre plus tard. De ce fait, les arcs ne vont que de la couche voiture ou vélo vers la couche piéton mais pas dans l'autre sens. Le cas de vélo en libre service et des taxis sera traité plus loin. Un exemple illustrant cela est visible dans la **figure 3.1**.

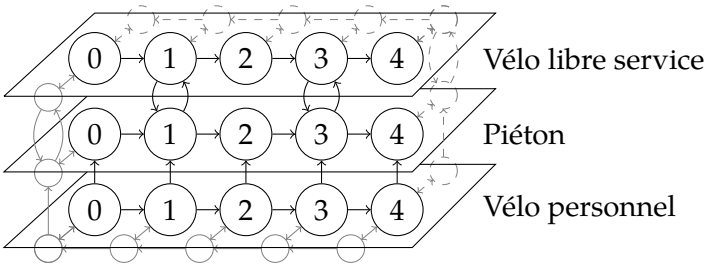


Figure 3.1 Exemples de graphes en plusieurs couches

3.2.2 Transports en commun

Les transports en commun se caractérisent par un nombre restreint de possibilités et surtout par des horaires de fonctionnement très précis. Nous considérons que les transports en commun sont représentés par une succession d'arrêts qui forment une ligne. Chaque ligne peut être parcourue régulièrement par des bus, métros, trains, etc. qui respectent scrupuleusement la même séquence d'arrêts. Cependant, rien n'empêche que le temps de parcours entre deux arrêts soit différent selon l'heure de la journée. Par exemple un bus pris dans la circulation mettra plus de temps en heure de pointe qu'en heure creuse.

Lorsque une même ligne propose plusieurs parcours différents (par exemple un parcours *omnibus* et un parcours *express* ou les branches d'une ligne de RER), nous considérons qu'il s'agit de deux lignes distinctes comme dans la **figure 3.2**. Cela permet donc de garantir la contrainte FIFO puisqu'aucune rame ne pourra en dépasser une autre sur un même arc.

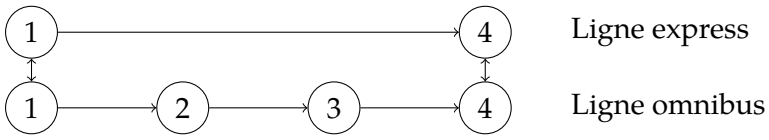


Figure 3.2 Exemple de séparation d'une ligne en deux lignes express et omnibus pour garantir la condition FIFO

Chaque arrêt est modélisé par un nœud et les nœuds sont reliés par des arcs suivant le tracé de la ligne des transports en commun. Un arrêt qui est parcouru par plusieurs lignes sera modélisé par un nœud par ligne comme montré dans la **figure 3.3**. En effet il est nécessaire de pouvoir modéliser le temps de changement entre les deux lignes (par exemple pour aller d'un quai à un autre).

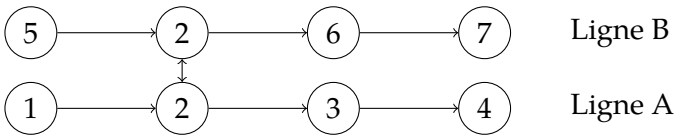


Figure 3.3 Exemple de la modélisation d'un arrêt en commun entre deux lignes

Afin de prendre en compte le temps d'attente dans une station, celui-ci est intégré dans les fonctions de coût associées à chaque arc. Ainsi, en arrivant à l'instant t au nœud u pour aller au nœud v, $f_{uv}^{t}(t)$ retournera l'instant le plus tôt auquel il sera possible d'arriver en v en empruntant cet arc et en prenant en compte ce temps d'attente.

Dans le cas d'une ligne de métro, la notion de ligne semble assez évidente. Par contre pour un réseau ferroviaire, il se peut qu'une ligne ne soit empruntée qu'une seule fois

par jour. Les conséquences d'une telle approche sur la taille du graphe est étudiée dans la **section 3.4 page 71**.

Afin de pouvoir modéliser le temps de changement entre deux lignes, les nœuds sont dédoublés pour chaque ligne. Le poids sur l'arc qui relie les deux nœuds dédoublés est le temps nécessaire pour effectuer le changement (par exemple aller d'un quai à un autre). Ce poids permet également de prendre en compte une marge de sécurité.

3.2.3 Vélos en libre service et taxis

Ces modes ont la particularité d'utiliser le réseau routier traditionnel, mais ne sont accessibles qu'à des endroits particuliers (borne de vélo, file de taxi).

Un réseau de taxi sera modélisé par le réseau routier complet. Il est possible de passer de la couche taxi à la couche piéton à tout nœud. Cependant passer de la couche piéton à la couche taxi n'est possible qu'en certains endroits prédéfinis comme des bornes à taxis. Il serait également possible de modéliser le fait qu'on peut appeler un taxi n'importe où. Le temps nécessaire pour passer de la couche piéton à la couche taxi dépendra de l'heure et la position géographique. Malheureusement pour ce niveau de détail il serait nécessaire d'avoir des statistiques précises sur le temps d'attente d'un taxi.

De même les vélos en libre service seront modélisés par une couche correspondant au réseau cyclable, c'est-à-dire tous les arcs que peuvent emprunter les vélos (le réseau routier sans les arcs interdits au vélo et avec les voies réservées au vélo). Les échanges avec la couche piéton ne se font qu'au niveau des bornes. Il peut être tentant de modéliser la couche comme une clique (graphe reliant chaque nœud à tous les autres). Cette approche peut se justifier dans les très petits réseaux. Par exemple celui de Perpignan ne comporte que 15 stations. La couche serait alors composée de $15^2 = 225$ arcs. Cependant à Paris, avec plus de mille stations, la couche comporterait plus d'arcs qu'en modélisant le réseau routier.

3.2.4 Co-voiturage et transport à la demande

Nous avons décidé de ne pas prendre en compte le co-voiturage et le transport à la demande. En effet, nous nous intéressons aux itinéraires de point à point pour une personne donnée. Dans le cas de ces modes de transport, il est nécessaire de combiner les trajets de plusieurs personnes. La difficulté est alors sur le plan de la modélisation des objectifs à optimiser et non pas sur les algorithmes utilisés ou la modélisation des modes.

Le lecteur s'intéressant à ces problématiques pourra lire Cordeau, [76] pour avoir un état de l'art sur différentes variantes du transport à la demande et Baldacci, [77] pour un exemple d'optimisation dans le cas du co-voiturage.

3.3 Nature des fonctions de coût

Nous avons vu dans la **section 1.3.4 page 26** qu'il peut être possible d'arriver à des chemins de moindre coût de longueur infinie dès que les coûts dépendent du temps. Étant donné que la notion de *dominance* au sens de Pareto ne définit pas un ordre total, il est important de définir les conditions que doivent remplir les fonctions de coût pour éviter des chemins de longueur infinie.

3.3.1 Formalisation des fonctions de coût

En considérant o coûts, il existe pour tout nœud u autant de coûts notés c_u^i avec i allant de 1 à o. L'ensemble des coûts associés à un nœud est le vecteur $\mathbf{c}_u \in \mathbb{R}^o$. Par la suite nous considérons que le premier élément du vecteur de coûts est le temps. Cela implique que le temps est nécessairement calculé.

Pour chaque arc (u, v), il existe o fonctions de coût notées $f_{uv}^i : \mathbb{R} \to \mathbb{R}$. Chaque fonction est fonction du temps. L'ensemble des fonctions de coût forme le vecteur $\mathbf{f}_{uv} : \mathbb{R} \to \mathbb{R}^o$.

Le coût à un nœud v en arrivant du nœud u se calcule de la manière suivante (on considère que le premier élément des coûts est le temps) :

$$\mathbf{c}_v = \mathbf{c}_u + \mathbf{f}_{uv}(c_u^1)$$

3.3.2 Caractéristiques des fonctions de coût

Il est probable que la majorité des coûts soient proportionnels à la longueur de l'arc et indépendant du temps. En effet les émissions de CO_2 d'une voiture ne dépendent que de la distance parcourue, tout comme la mesure du nombre de changements.

La durée de parcours du temps est la fonction la plus variable. Elle modélise le temps d'attente du prochain train ainsi que les variations des durées de parcours sur un axe routier. Puisque le temps de parcours d'un arc est toujours positif — à moins d'être capable de remonter dans le temps — traverser un arc plus tard fera toujours arriver plus tard. Il s'agit là de la condition FIFO que nous avons tenu à maintenir dans notre modèle.

Cependant, dans le cas de fonctions de coût, la situation est plus compliquée. En effet, même si tous les coûts sont positifs, un chemin optimal peut être de longueur infinie. Nous avons déjà souligné que pour qu'il existe un chemin optimal de longueur finie, une condition suffisante est qu'à partir d'un certain rang, le coût ait une borne inférieure Orda,Rom [25]. Cette propriété est intéressante dans le cas d'une optimisation monoobjectif. Dans le cadre multiobjectif, elle n'est plus satisfaisante.

Il est important de faire attention au cas difficile de fonctions de coût continues décroissantes (qui génère un front de Pareto continu). Ce genre de situation est relativement fréquent : péage urbain (vu dans le chapitre précédent), les places de parking sont souvent gratuites à partir d'une certaine heure, ou bien le prix d'une place de deux trains effectuant exactement le même trajet sera différent selon l'heure.

Sachant que nous ne souhaitons pas devoir considérer le temps d'attente à un nœud comme un paramètre, nous divisons les fonctions de coût en deux ensembles : les fonctions de coût continues décroissantes et les fonctions dont les coûts sont croissants par morceaux.

Fonctions de coût continues et décroissantes

Lorsque qu'il existe un intervalle de temps pendant lequel la fonction de coût est décroissante et continue, alors on se retrouve dans la situation décrite avec le péage urbain. Il existe une infinité d'instant de passages Pareto-optimal et il existe donc une infinité de chemins Pareto-optimaux.

Nous faisons le choix de refuser ce type de fonctions de coût. Si on accepte une approximation, il est possible de discrétiser le temps pour se ramener au deuxième cas. Cela va à l'encontre de principes que nous nous étions fixé initialement (à savoir aucune approximation inutile). Cependant nous pensons que pour une application réelle, il n'existe pas de telle situation. De plus ce type de fonction de coût provoquant un nombre infini de chemin Pareto-optimaux, le problème est plus profond que la simple modélisation du graphe.

Fonctions de coût non-décroissantes par morceaux

En pratique, les fonctions de coût ne sont pratiquement jamais décroissantes et continues. En effet, les tarifs de parking ou de péages seront découpés en tranches horaires. À

l'intérieur de chaque tranche horaire, les coûts sont constants ou croissants. Il en est de même pour les trains où chaque tranche serait l'intervalle entre les deux trains.

3.3.3 Implémentation des fonctions

Les fonctions de coût liés à des transports en commun peuvent se modéliser comme un simple tableau de paires (départ-arrivée). En ayant l'heure d'arrivée au nœud de départ, il suffit de parcourir la liste pour trouver le prochain départ et donc l'heure d'arrivée à la fin de l'arc. Afin d'accélérer la recherche, une recherche par dichotomie peut être effectuée.

Dans le cas de fonctions de coût arbitraire, il est évidemment nécessaire d'adapter une implémentation au cas par cas. La plupart des publications existantes considèrent des fonctions linéaires par morceaux. En effet, chaque segment ne nécessite que deux points pour être représenté. De plus cela permet de facilement calculer des combinaisons (par exemple pour la création de raccourcis dans les algorithmes basés sur des contractions présentés dans la **section 1.2.3.4**). De plus les bornes inférieures et supérieures sont également faciles à calculer pour une utilisation heuristique. Enfin, cette modélisation de coût arbitraire est à notre avis suffisamment générique et convient à la majorité des besoins.

Parfois l'horaire de passage n'est pas connu avec précision : seule une fréquence est donnée (par exemple le métro de Toulouse circule toutes les 3 minutes en heure de pointe). Dans ce cas là, il suffit de prendre le temps maximal d'attente. Il est en effet préférable de laisser quelques minutes de marge que prendre le risque de rater une correspondance.

3.3.4 Approche statistique des coûts

Dans la réalité, les horaires d'autobus ou l'état du trafic sur un périphérique à l'heure de pointe ne peuvent pas être connus avec une précision infinie. Il serait donc intéressant de pouvoir prendre en compte ces incertitudes. Plusieurs approches sont possibles. Il est possible de considérer :

- une marge constante de sécurité (pour les transports en commun) ;
- un temps moyen selon l'heure (pour le transport en voiture) ;
- plusieurs temps de parcours et calculer plusieurs chemins ;
- un objectif supplémentaire modélisant le risque ;

– des fonctions stochastiques.

Les trois premiers cas peuvent s'adapter facilement (encore faut-il avoir aux accès données, comme par exemples celles issues des boucles de détections installées sur les grands axes !) au modèle proposé. Une approche stochastique nécessite une étude bien plus poussée qui sort du cadre de cette thèse.

3.4 Considérations pratiques

3.4.1 Taille du graphe généré

Le modèle proposé a tendance à multiplier les nœuds et les arcs. Le but de cette section est d'évaluer la taille du réseau généré afin de savoir dans quelle mesure il peut être mis en application.

En considérant que le réseau routier « réel » comporte n nœuds et m arcs, un graphe multimodal comportant une couche voiture, piéton, vélo, vélo en libre service, taxi et transports en commun, comportera donc $4n$ nœuds et $4m$ arcs pour modéliser les 4 premières couches. En considérant que toutes les couches sont reliées à la couche piéton par un arc dans chaque sens, il existe $2 \cdot 3n$ arcs de liaison.

En ce qui concerne les transports en commun, l'estimation du nombre de nœuds et d'arcs générés est difficile. Dans Paris et sa banlieue, la RATP exploite environ 350 lignes de bus pour 5 000 arrêts[11]. En considérant que chaque ligne est composée d'un peu moins de 30 arrêts (plusieurs lignes pouvant partager les mêmes arrêts), le modèle proposé nécessitera 10 000 nœuds et 20 000 arcs pour modéliser le réseau de bus de la RATP. En prenant en compte les autres modes de transport en commun moins denses que le réseau de bus et sachant que la Paris et sa banlieue représentent environ 100 000 nœuds, la couche de transports en commun aura au plus n nœuds et il y aura au plus n arcs (arcs de changements de mode compris).

On en déduit donc que le graphe multimodal d'une région comportant n nœuds et m arcs dans le réseau routier sera modélisée par au plus $5n$ nœuds et $4m + 7n$ arcs pour les cinq modes de transport considérés. Sachant qu'en moyenne sur un réseau routier on a $m \approx 4n$, le graphe multimodal comportera donc au plus $23n$ arcs.

[11] Source : Syndicat des transports d'Île-de-France, `http://www.stif.info`

3.4.2 Occupation mémoire

En utilisant une liste d'adjacence, il est possible de modéliser un graphe de n nœuds et m arcs de manière à ce qu'il occupe $8n + 4km$ octets en mémoire où k est la taille nécessaire pour stocker toutes les données relatives aux fonctions de coût associées à chaque arc. Étant donné que la représentation en mémoire de chaque fonction de coût dépend fortement de leur nature et du nombre d'objectifs considéré, estimer k est particulièrement difficile. Cela peut aller de un octet (pour exprimer le temps de parcours en secondes) à facilement un kilo-octet (tous les horaires de passage d'un métro dans la journée[12]).

Sachant qu'avec quatre modes de transports, nous avons $m \approx 23n$, on peut majorer la consommation de mémoire par $23{\times}4kn \approx 100kn$. Dans le cas de Paris et de sa banlieue ($n = 100\ 000$), pour $k = 1\ 000$, la consommation de mémoire est de 10 Go. Cela est acceptable pour un serveur haut de gamme actuel. Cependant le passage à une plus grande échelle devient délicat. La France est en effet représentée par un million de nœuds et l'Europe par un peu moins de vingt millions de nœuds. Si l'information moyenne à stocker sur chaque arc est de $k = 100$, il est possible de représenter simultanément toute la France. Il pourrait être tentant d'avoir une approche hiérarchique et calculer localement les trajets courte distance et ignorer certains modes sur de grandes distances en ne gardant par exemple que les trains longue distance entre deux villes éloignées. Pourtant pour aller de Paris à Toulouse il est possible de prendre le TGV à la gare de Montparnasse ou le Teoz à la gare d'Austerlitz. L'écart de temps entre les deux trajets est de seulement une heure. De ce fait, selon l'attente, les connexions avec les trains de banlieue, prendre le train le plus lent peut se révéler être plus rapide. Ce genre de situations seraient difficiles à prendre en compte en découpant le calcul d'itinéraire.

Un passage à l'échelle nécessiterait une étude plus fine de la mémoire nécessaire (en pratique un nombre très significatif des coûts sont des constantes). De plus les fonctions de coût les plus « lourdes » sont celles des transports en commun qui sont nettement moins denses en province qu'en région Parisienne.

Il pourrait être intéressant d'étudier une approche pour ne pas avoir besoin de stocker tout le graphe multimodal en mémoire. Ces deux points n'ont cependant pas été étudiés dans le cadre de cette thèse.

La forte consommation de mémoire de l'approche multicouche est cependant à modérer. En effet, 8 octets suffisent pour représenter l'existence d'un arc. Par contre les fonctions

[12] La ligne circulaire Yamanote de Tokyo est la plus chargée au monde avec 3,5 Millions de passagers par jour. Les rames y circulent 21h par jour, avec en pointe 25 rames par heure.

de coût associées à cet arc peuvent représenter plus de 1 000 octets. La consommation de mémoire sera donc importante même dans le cadre d'une représentation multivaluée si l'on désire avoir des fonctions de coût dépendantes du temps. C'est donc le fait de prendre en compte des grilles horaires et non pas le modèle multi-couches qui est la cause de la consommation en mémoire.

3.4.3 Obtention des données

L'obtention des données a été longtemps une tache difficile. En effet les données cartographiques étaient vendues par un nombre restreint d'éditeurs. De plus les données sont souvent très orientés vers le transport routier et posent généralement peu d'informations pour un réseau multimodal (présence de bandes cyclables par exemple[13].

En ce qui concerne les données de transports en commun, celles-ci sont encore considérées en France comme ayant une importance stratégique. Il n'est donc pas possible de les récupérer sans un accord particulièrement restrictif.

L'initiative *OpenStreetMap*[14] consiste à créer une base de données géographique communautaire sur un mode de fonctionnement analogue à celui de Wikipedia. Les données sont globalement de très bonne qualité (en particulier sur les grandes villes) et permettent d'avoir des informations généralement absentes des cartes commerciales telles que les voies réservées aux piétons ou les aménagements cyclables.

En ce qui concerne les données des transports en commun, un certain nombre[15] de compagnies de transports proposent leurs horaires au format GTFS (Google Transit Feed Specification) dans le but de développer des applications autour de leurs réseaux de transport.

Il est regrettable que — à notre connaissance — aucun réseau en France ne diffuse ses horaires dans un format électronique libre.

L'unique exception est la ville de Rennes qui a commencé à ouvrir ses premières données au printemps 2010 en commençant par l'état des stations de vélo. À terme les horaires des bus et d'autres informations concernant l'agglomération seront rendues publiques à la fin de l'été 2010[16].

---

[13] D'autant plus qu'elles évoluent très rapidement. Par exemple la généralisation en juillet 2010 des contresens cyclables à toutes les voix limitées à 30km/h modifie considérablement les itinéraires des cyclistes

[14] `http://www.openstreetmap.org`

[15] `http://www.gtfs-data-exchange.com/`

[16] `http://data.keolis-rennes.com/`

Conclusion

Nous avons proposé un modèle pour représenter un réseau multimodal. Ce modèle permet de facilement prendre en compte un grand nombre de modes de transport (la seule exception notable étant le covoiturage) et cela de manière très naturelle. Le graphe obtenu a la particularité de respecter la contrainte FIFO, ce qui permet d'employer des algorithmes traditionnels directement. La contrepartie est un graphe de grande taille. L'utilisation sur une grande ville ne pose pas de problème (par exemple à l'échelle de l'Île-de-France), mais peut devenir problématique sur une plus grande échelle si un grand nombre de modes sont considérés.

Les deux chapitres suivants vont se pencher sur l'application des algorithmes sur ce modèle. Nous nous nous intéressons tout d'abord à l'itinéraire le plus rapide pour tester une application directe.

Chapitre 4
Itinéraire multimodal le plus rapide

No stop signs, speed limit, Nobody's gonna slow me down

— AC/DC, Highway To Hell

Introduction

Afin de tester une application directe du modèle proposé dans le chapitre précédent, nous proposons de calculer le chemin le plus *rapide*. Les algorithmes proposés ne sont pas nouveaux et ont surtout pour but de démontrer que le modèle permet de fonctionner sur des régions nettement plus grandes que celles considérées jusqu'à présent.

Cela permet aussi de mieux comprendre quelle est la conséquence d'utiliser des fonctions de coût à la place de coûts constants et de tester des heuristiques simples telles que celles utilisées dans l'algorithme A*.

4.1 Algorithmes

4.1.1 Dijkstra généralisé

Étant donné que le modèle du réseau multimodal a été pensé afin de pouvoir utiliser directement des algorithmes existants, il n'existe pas d'adaptation à faire à l'algorithme de Dijkstra dépendant du temps.

4.1.2 Algorithme A*

Afin d'améliorer sensiblement les performances, nous avons adapté l'algorithme heuristique A* au cas dépendant du temps. L'heuristique mesurant la distance à la cible est moins triviale dans ce cas. Il s'agit en effet d'estimer au mieux le temps nécessaire pour

atteindre la destination, sans pour autant le surestimer. Une borne sûre est le temps qui serait nécessaire pour parcourir la distance à vol d'oiseau avec le mode de transport le plus rapide.

Cependant dans le cas du transport multimodal, l'écart entre cette heuristique et la durée réelle risque d'être particulièrement importante (trajet en vol d'oiseau en TGV par rapport à un trajet avec des détours à pieds). À cause de cet écart entre l'heuristique et la solution optimale, il est difficile de savoir si cette heuristique engendrera des gains significatifs.

4.2 Expérimentations

4.2.1 Implémentation

Le graphe a été modélisé grâce à la bibliothèque *Boost Graph Library* qui implémente également les algorithmes de Dijkstra et A*.

Cela montre que, comme nous l'avions souhaité, il est possible d'utiliser directement des implémentations d'algorithmes existants.

Les calculs ont été effectués sur un simple ordinateur portable ayant un core 2 duo à 1,67 Ghz avec 2 Go de mémoire vive ayant suffit pour toutes les expérimentations. Un seul cœur a été utilisé.

4.2.2 Ensemble de tests

Le premier ensemble de test est basé sur les données de la ville de San Francisco. Il comprend le réseau routier, le réseau Muni[17](bus et tramway), le réseau Bart[18] (train de banlieue) et le réseau cyclable.

Le deuxième ensemble de test est basé sur Los Angeles. Le réseau de transports en commun est le réseau Metro[19].

Ce tableau permet de comparer les deux instances en fonction de la région couverte, le nombre de nœuds et d'arcs sur la couche routière ainsi que le nombre d'arrêts de tous les transports en commun confondus.

[17] `http://www.sfmta.com/`

[18] `http://www.bart.gov/`

[19] `http://www.metro.net/`

Ville	Surface	Population	Nœuds	Arcs	Arrêts
San Francisco	100km^2	744 000	14 529	43 788	3 659
Los Angeles	3000km^2	5 000 000	135 663	375 382	15 565

4.2.3 Protocole d'expérimentation

Voici la liste des points que nous souhaitons comparer :

- le surcoût lié à l'utilisation de fonctions de coûts au lieu de coûts constants ;
- l'influence du nombre de modes de transport sur le temps de calcul ;
- l'efficacité des heuristiques :
  - arrêter la recherche lorsque la destination est atteinte,
  - algorithme A* ;
- l'impact de la proximité du nœud de départ et d'arrivée.

Le temps de calcul indiqué est le temps moyen calculé sur 100 calculs d'itinéraires. Avec trois modes, le nœud de départ est choisi aléatoirement dans la couche vélo et le point d'arrivée dans la couche piéton. Le scénario est donc celui d'une personne quittant son domicile à vélo, mais pouvant le déposer (pour prendre les transports en commun par exemple). Avec deux modes, le nœud de départ et d'arrivée sont dans la couche piéton.

Afin de pouvoir comparer l'influence de la distance à vol d'oiseau entre le départ et l'arrivée et le temps de calcul, nous regroupons les tests en *classes de distances*. Le nombre et la taille des classes varient en fonction de la taille de la ville.

4.2.4 Les paramètres testés

En ce qui concerne la distance entre le départ et l'arrivée de chaque classe de distance, pour San Francisco nous avons considéré les intervalles [0 ; 2km ; 4km ; 6km ; 8km]. Pour les classes de Los Angeles, les intervalles sont [0 ; 3km ; 5km ; 10km ; 15km ; 20km].

Les modes testés sont :

- 1 mode : marche ou transport en commun ;

– 2 modes : marche et transport en commun ;

– 3 modes : vélo, marche et transport en commun.

Les algorithmes employés sont :

– par défaut : Dijkstra tronqué dès que le nœud d'arrivée est trouvé ;

– complet : calcule le plus court chemin d'un nœud vers *tous* les nœuds ;

– A*.

4.3 Résultats expérimentaux

La **figure 4.1** et la **figure 4.2** représentent le temps de calcul selon les différentes classes de distance et les différents tests effectués.

4.3.1 Utilisation de fonctions de coût

On constate un léger surcoût lorsque des fonctions de coût sont utilisées au lieu de coûts constants (Graphes 3 et 4). Cependant dans les deux instances de test avec de la marche, que ce soit la distance avec un algorithme traditionnel ou la durée en utilisant des fonctions de coûts, les temps de calcul sont tellement faibles qu'il est difficile d'en tirer des conclusions.

Lorsque l'on s'intéresse uniquement à un itinéraire avec des transports en commun dans le Graphe 1 (donc où tous les coûts sont variables), on constate que le temps de calcul est considérablement plus long, ce qui est prévisible étant donné qu'il est nécessaire à chaque évaluation d'un arc de chercher l'horaire du prochain bus dans une table. Dans le cas de Los Angeles, sur la classe où les départs et arrivées sont le plus espacés (plus de 15 km), les transports en communs purs ne sont pas pris en compte car il n'existe pas de paires de points suffisamment éloignées desservies.

Il est beaucoup plus étonnant que lorsque l'on ne considère que les transports en communs, le temps de calcul soit plus long que lorsqu'on l'on considère les transports en commun *et* les autres modes (Graphes 1, 5 et 6). Une explication possible est que du fait de la topologie des réseaux de transports (par exemple avec des lignes express qui traversent une grande partie de la ville sans arrêt ou encore le besoin de faire des changements de ligne), pratiquement tout le réseau doit être parcouru avant d'arriver à destination. On

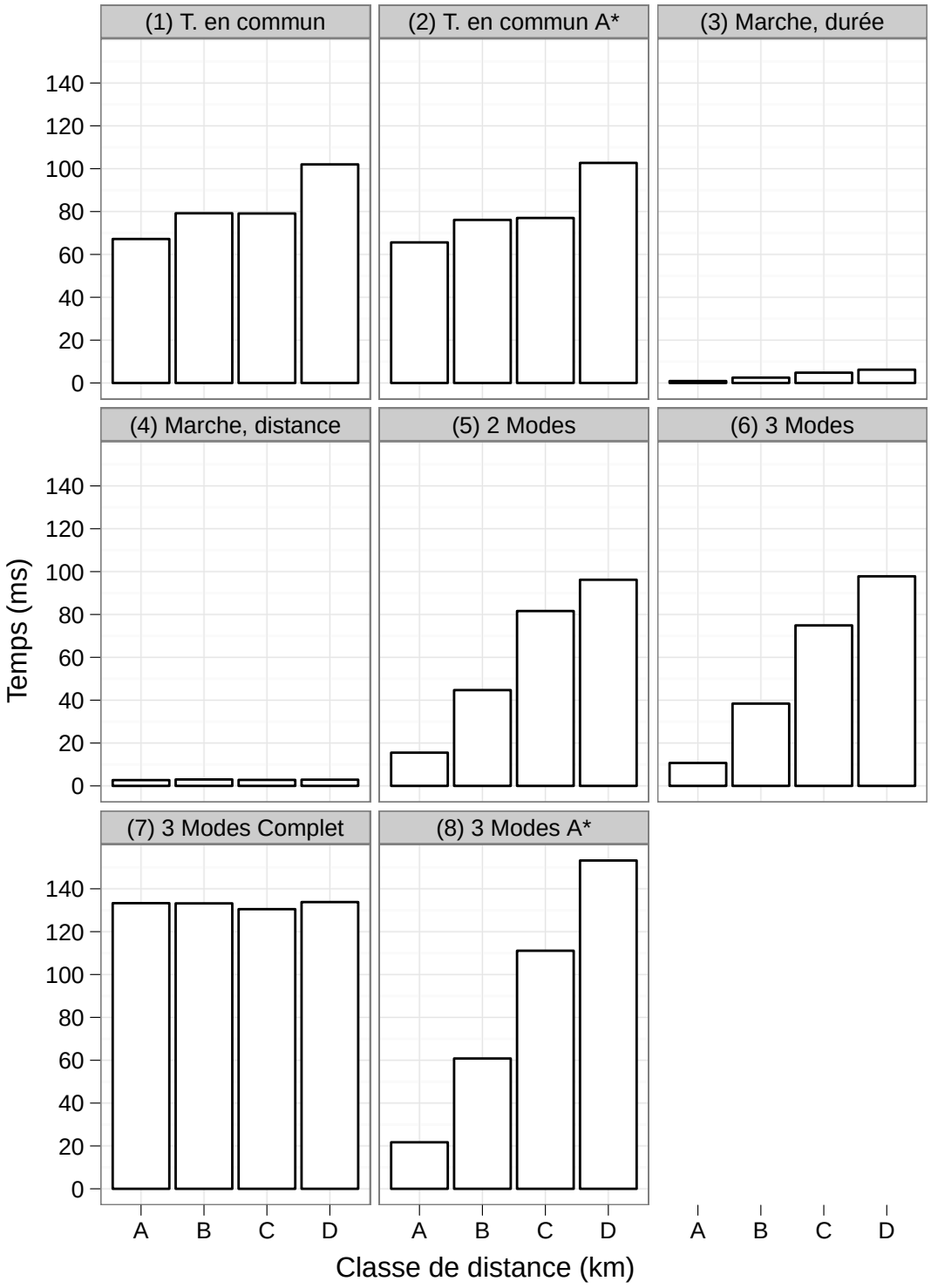


Figure 4.1 Temps de calcul selon les heuristiques, les modes de transport et la distance (San Francisco)

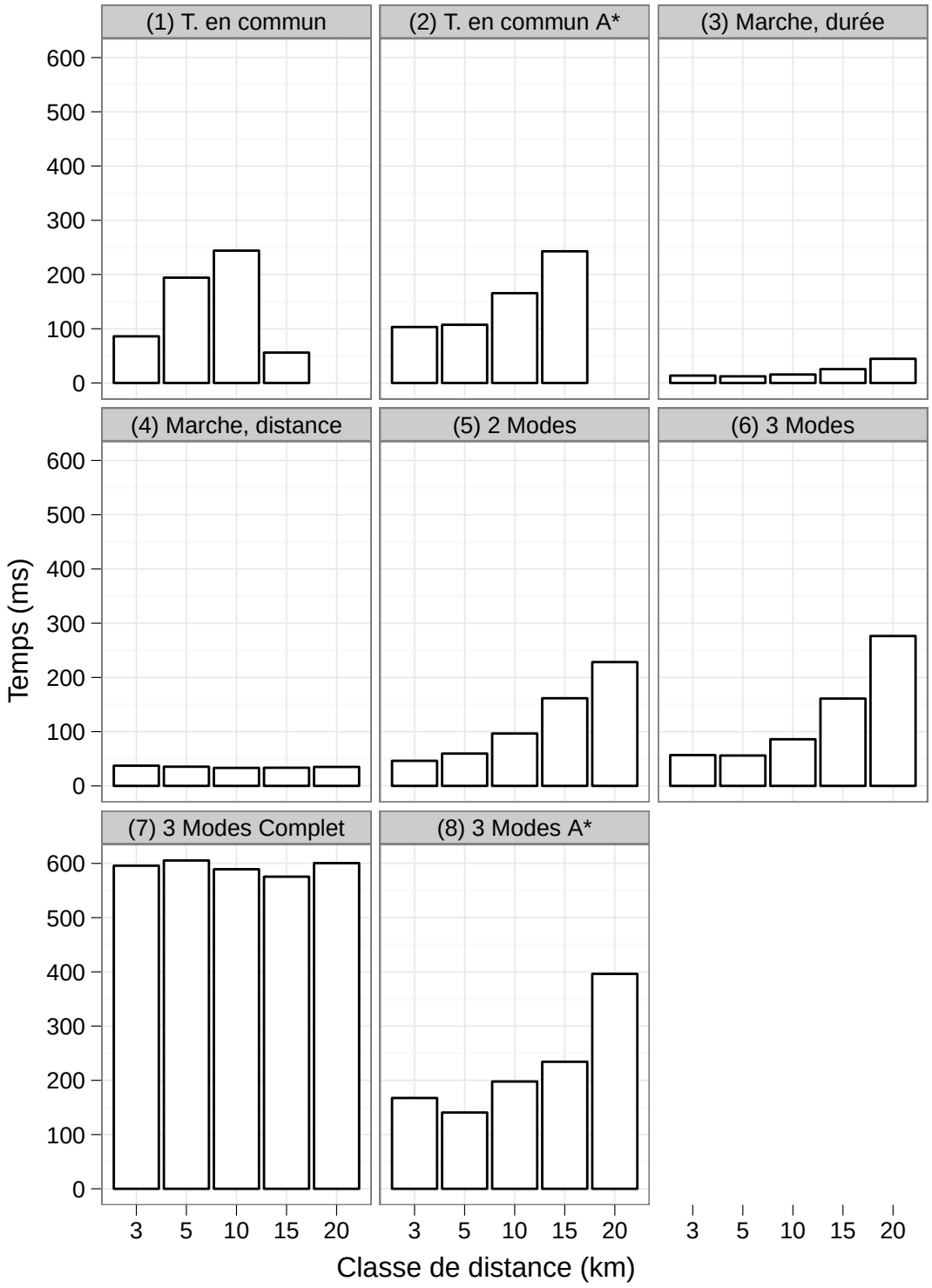


Figure 4.2 Temps de calcul selon les heuristiques, les modes de transport et la distance (Los Angeles)

constate de plus que pour la classe où les distances sont les plus grandes, les temps sont très similaires confirmant cette hypothèse.

4.3.2 Taille des instances

Il est bien évidemment difficile à partir de ces deux ensembles de tests de prédire le comportement en fonction de la taille du réseau. Dans le pire cas (calcul de Dijkstra complet, graphe 7), l'instance de Los Angeles nécessite cinq fois plus de temps. Le graphe est pratiquement dix fois plus grand (en terme de nœuds et d'arcs), mais ne comporte que cinq fois plus d'arrêts de transports en commun. Il n'est pas étonnant donc que ce soit la taille du réseau de transports en commun qui détermine le temps de calcul nécessaire plus que la taille du réseau routier.

On constate aussi que pour des classes de distance comparables (moins de 10 km), les deux instances ont des temps de calcul comparables. Une heuristique aussi simple que couper la recherche dès que le nœud d'arrivée est atteint permet donc de calculer efficacement une route courte par rapport à l'ensemble du graphe.

Il faut aussi noter que la plus grande instance qui couvre une zone de taille particulièrement importante nécessite au pire 600 ms pour calculer le plus court chemin d'un seul nœud vers tous les autres. En pratique le temps de calcul est plutôt de l'ordre de 200 ms pour un trajet entre 15 et 20 km. Cette approche peut donc être utilisée sans le moindre problème sur une très grande agglomération.

4.3.3 Algorithme A*

À première vue, il peut sembler étonnant que l'algorithme heuristique A* ne permette pas de réduire le temps de calcul (Graphes 1 et 2 ou Graphes 6 et 8). Le nombre de nœuds visités est diminué sensiblement, mais le gain de temps lié à cette réduction est annulé par le temps nécessaire pour évaluer l'heuristique. Un problème particulier au transport multimodal est qu'il est impossible d'avoir une borne inférieure de bonne qualité si on désire rester optimal à cause du grand écart de vitesse entre les modes de transport. C'est pour cela que le nombre de nœuds visités ne peut pas être autant réduit que dans le cas du plus court chemin statique.

Conclusion

Le calcul d'itinéraire multimodal le plus rapide est très simple à mettre en œuvre et permet d'être utilisé sans problème de performance sur une agglomération aussi grande que Los Angeles.

L'algorithme A* se révèle ne pas être une approche pertinente. En revanche, tronquer la recherche dès que le nœud d'arrivée est visité est une heuristique très simple qui se révèle être très efficace pour le calcul d'un trajet court à l'intérieur d'une grande région. Il est donc possible de considérer des régions bien plus grandes si dans la majorité des cas, les itinéraires demandés sont relativement courts (quelques dizaines de kilomètres).

Cependant, avoir l'itinéraire le plus rapide n'est pas toujours le plus pertinent. Afin de proposer des itinéraires plus intéressants à l'utilisateur, nous nous penchons dans le chapitre suivant sur une approche multiobjectif qui est au cœur de cette thèse.

Figure 4.3 Exemple de chemin le plus rapide à Rennes (marche et vélo libre service)

Figure 4.4 Exemple de chemin le plus rapide à San Francisco (marche, tramway, marche, bus et marche)

Chapitre 5
Meilleur itinéraire multimodal multiobjectif

Il n'y a que les imbéciles qui ne changent pas d'avis. C'est mon avis. Et je ne vois pas pourquoi j'en changerais
— Le Chat, Philippe Geluck

Introduction

Ce chapitre porte sur l'extension multiobjectif du calcul d'itinéraire et constitue donc le cœur de cette thèse. Le chapitre 2 a présenté l'intérêt d'une telle approche pour l'utilisateur et nous nous focaliseront sur les algorithmes et les résultats expérimentaux.

En se basant sur la modélisation du graphe multimodal présentée dans le troisième chapitre, nous proposons plusieurs algorithmes pour calculer le plus court chemin multiobjectif. La première approche est l'extension directe de l'algorithme de Martins. Les deux autres approches sont basées sur l'algorithme de Tsaggouris et sur une forme particulière de contraction de graphe, les *contraction hierarchies*.

Nous comparons également les résultats selon différents objectifs considérés afin de vérifier si le comportement est cohérent avec les observations du chapitre 2.

5.1 Algorithme de Martins dépendant du temps

L'algorithme de Martins est l'extension directe de l'algorithme de Dijkstra en multiobjectif. C'est donc la première approche qui a été testée pour le calcul d'itinéraire multimodal multiobjectif. Il était nécessaire de l'adapter afin de gérer la dépendance au temps.

Nous avons posé les conditions dans le chapitre 2 sur les fonctions de coût pour qu'il soit possible de calculer le front de Pareto complet.

L'adaptation de l'algorithme de Martins à la dépendance du temps est simples. Il suffit en effet de modifier la manière dont les coûts d'une nouvelle étiquette sont calculés. Cependant il est nécessaire de faire attention à la nature des fonctions de coût pour être sur que l'algorithme finira.

```
1  FONCTION Martins(source, nœuds, arcs, couts)
2      Q := (source, 0, null)
3      P := {}
4      TANT QUE Q est non vide FAIRE
5          (u, c, p) := étiquette de Q minimisant O
6          Q := Q \ {(u, c, p)}
7          P := P ∪ {(u, c, p)}
8          POUR TOUT v successeur de u FAIRE
9              POUR TOUT objectif i FAIRE
10                 c2_i := c_i + f^i_uv(c_0)
11             Q := Q ∪{(v, c2, u)}
12         ÉLIMINER ÉTIQUETTES DOMINÉS(Q)
```

Algorithme 5.1 Algorithme de Martins généralisé (dépendant du temps)

Les lignes 9 et 10 dans **l'algorithme 5.1** définissent la manière de calculer le coût **c2** de la nouvelle étiquette en fonction de l'instant c_0 de l'étiquette précédente.

5.1.1 Heuristiques

Il n'est pas possible de tronquer la recherche dès qu'un chemin est trouvé comme dans la recherche à un objectif unique. En effet, il se peut qu'il existe un autre chemin non dominé et il est donc nécessaire de continuer la recherche.

Afin d'améliorer les temps de calcul, nous utilisons deux heuristiques simples à mettre en application.

Tester la dominance avec les chemins trouvés

À chaque création d'une nouvelle étiquette, on teste si elle est dominée par un des chemins trouvés. Si cela est le cas, étant donné que tous les coûts sont positifs, un trajet passant par cette étiquette ne pourra pas être non-dominé. Cette étiquette pourra donc être directement éliminée.

Dominance relâchée

Nous utilisons l'approche classique de dominance relâchée pour améliorer encore un peu les temps de calculs. En éliminant au plus tôt des solutions similaires, moins d'étiquettes sont générées et le temps de calcul devrait être réduit.

Ce test de domination nécessite de fixer combien de temps on est prêt à sacrifier pour économiser une unité d'un autre objectif. Par exemple, pour gagner un changement, on accepte un trajet dix minutes plus long ou encore pour économiser dix euros on accepte de faire un détour de 20 minutes.

Cette heuristique permet de modéliser une certaine indifférence de l'utilisateur entre deux choix très proches.

Lorsque deux éléments sont comparés avec la dominance relâchée, il est important de conserver la solution arrivant le plus tôt afin de ne rater aucune correspondance. Étant donné que les coûts sont croissants en fonction du temps (ou décroissants en un nombre finis de points traitables au cas par cas), ne pas garder le coût le plus tardif ne pose pas de problème d'optimalité.

5.1.2 Expérimentations

Instances

Les ensembles de test sur lesquels nous effectuons les tests sont les mêmes que celles présentées dans le chapitre précédent (San Francisco et Los Angeles séparés en choissant le départ et l'arrivée dans des classes de distances allant de 2 km à 20 km).

Nous considérons les objectifs suivants :

- durée (tps) ;
- changements de mode (mode), par exemple passer de la marche au bus ;
- changements de ligne (ligne), par exemple changer entre deux lignes de métro ;
- dénivelé positif (d+), uniquement à vélo ;

Dans toutes les expériences nous considérons trois modes : marche, transport en commun et vélo personnel. Un itinéraire commence à vélo et se finit à pied. Il est possible de

déposer le vélo à n'importe quelle station de transports en commun, mais il n'est plus possible de le réutiliser par la suite.

Les calculs ont été effectués sur un ordinateur portable ayant un core 2 duo à 1,67 Ghz et 2 Go de mémoire vive. Un seul cœur a été utilisé. L'implémentation est détaillée dans l'annexe A.

Protocole

Nous comparons différentes combinaisons d'objectifs avec à chaque fois les différentes combinaisons d'heuristiques. Nous utilisons également le même découpage en classes de distances que dans le chapitre 4. Chaque fois nous effectuons 10 calculs d'itinéraires et présentons les temps moyens.

5.1.3 Résultats

La **figure 5.1** indique le temps de calcul sur la ville de San Francisco tandis que la **figure 5.2** indique le nombre de solutions Pareto optimales trouvées. Les figures **5.3** et **5.4** correspondent aux mêmes résultats sur Los Angeles.

Nombre de solutions trouvées

Comme nous l'avions montré dans le chapitre 2, le fait d'utiliser des variables binaires (nombre de changements) génère bien moins de solutions Pareto optimales qu'avec des fonctions continues (dénivelé positif). Ainsi, considérer trois objectifs peut se révéler être plus facile que de n'en considérer que deux.

On constate qu'en pratique, considérer uniquement le nombre de changements de mode ou de lignes avec un départ ne génère que très peu de solutions non dominées. En effet, souvent le trajet avec le moins de changements est également le plus rapide.

Efficacité des heuristiques

Le temps de calcul est fortement influencé selon les instances. De l'ordre de 100 ms pour des objectifs faciles sur San Francisco, il atteint 14 secondes dans le cas le plus difficile. On constate que le fait de couper la recherche permet de contenir le temps de calcul lorsque les nœuds de départ et d'arrivée sont proches.

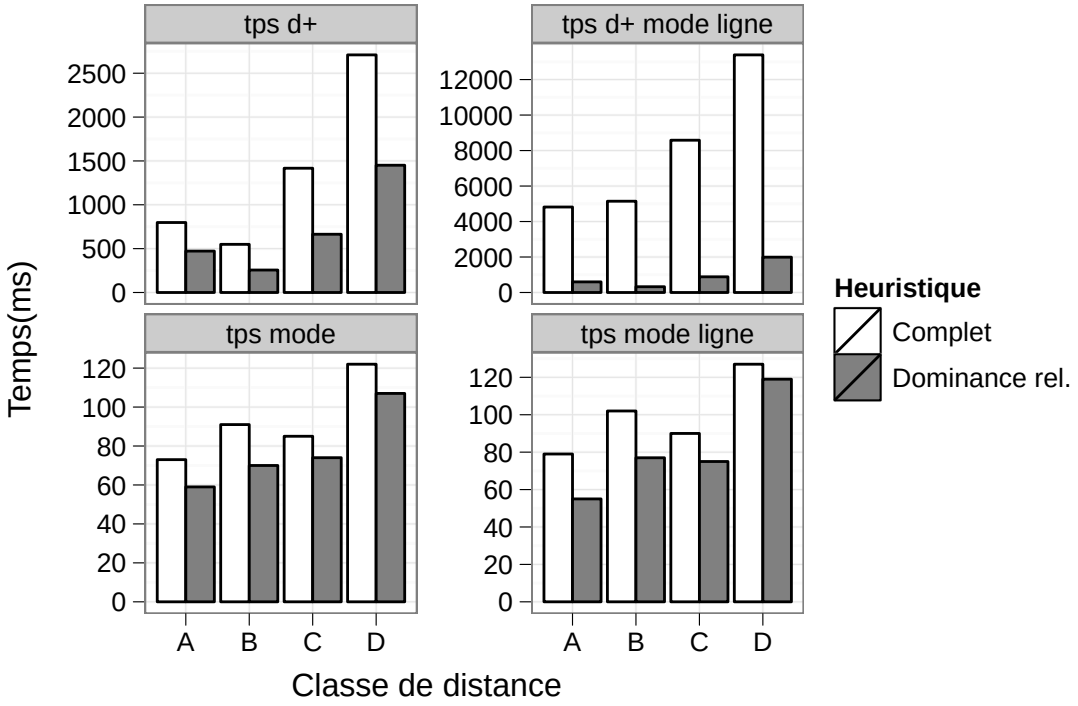


Figure 5.1 Temps de calcul selon les objectifs et les heuristiques (San Francisco)

La dominance relâchée permet de significativement diminuer le temps de calcul (plus d'un facteur 10 sur les cas les plus difficiles) et nous permet de rester sous la barre de la seconde pour le calcul d'itinéraire sur les instances de San Francisco. Sur les instances de Los Angeles, le gain est parfois encore plus significatif (facteur 50).

Efficacité à grande échelle

Les instances sur Los Angeles montrent les limites de cette implémentation. En effet, certains tests nécessitent plusieurs secondes voire ne sont pas calculables dans des temps raisonnables (plus de trois minutes). De ce fait il n'est pas possible d'avoir le front de Pareto complet dans un temps raisonnable sur les grandes instances et des objectifs difficiles. Cependant, en réduisant le nombre de solutions trouvées avec la dominance relâchée, ou en ne cherchant que des objectifs faciles, alors le temps de calcul reste acceptable.

Ces limites ne sont pas particulièrement gênantes par rapport à ce que nous nous étions fixés. En effet l'agglomération de Los Angeles fait partie des plus grandes du monde. De plus les calculs ont été fait sur un ordinateur portable. L'augmentation de la puissance des

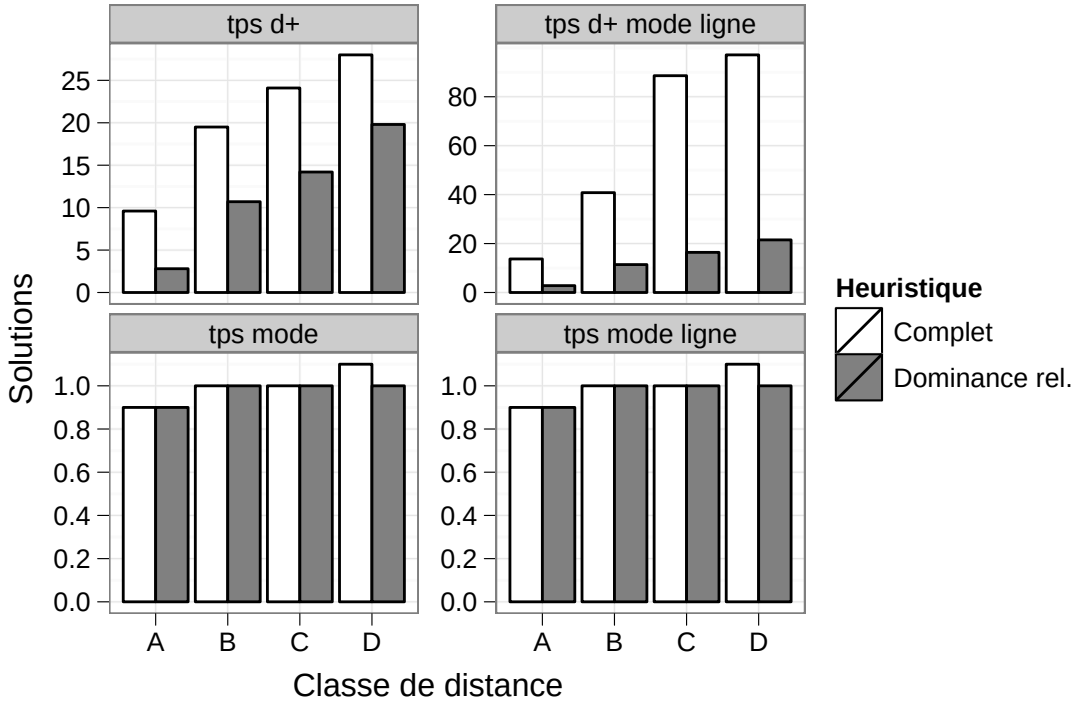


Figure 5.2 Nombre de solutions selon les objectifs et les heuristiques (San Francisco)

ordinateurs combinée à une implémentation plus optimisée devrait permettre de diviser plusieurs fois le temps de calcul.

5.1.4 Comparaison avec l'existant

Il est difficile de comparer ces résultat aux autres approches existantes du fait des champs d'application très différents. Les travaux les plus proches des nôtres sont ceux de Fallou Gueye Gueye, [75] qui optimise le temps de parcours et le nombre de changements sur Toulouse. La taille des régions testées ainsi que la taille des zones considérées sont détailles dans le **tableau 5.1**.

Notre approche est clairement plus performante en terme de temps de calcul, malgré un processeur moins puissant. De plus elle est plus simple à mettre en œuvre et permet de prendre en compte plus d'objectifs et pas seulement le temps et le nombre de changements.

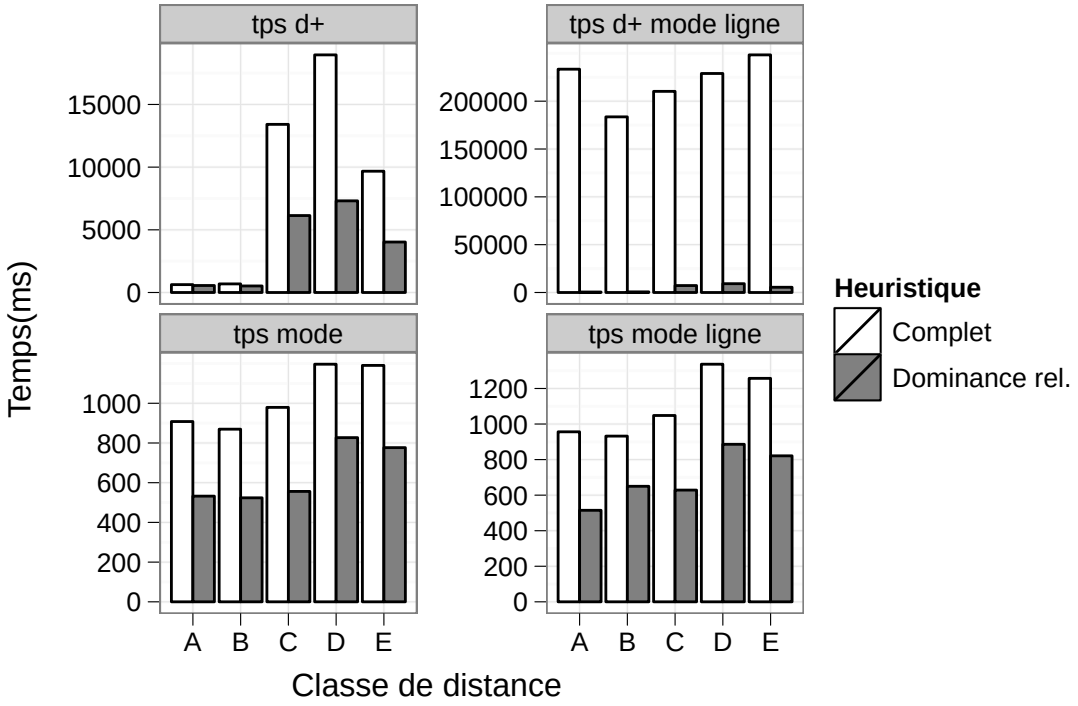


Figure 5.3 Temps de calcul selon les objectifs et les heuristiques (Los Angeles)

5.1.5 Conclusion

L'algorithme de Martins dépendant du temps a très facilement été adapté et fonctionne très bien sur notre modèle. Dans la plupart des cas, le temps de calcul est dans les limites que nous nous étions fixés. Cependant avec certains types d'objectifs le temps de calcul devient trop long (cela va de pair avec une forte augmentation du nombre d'éléments dans le front de Pareto) sur les instances les plus difficiles.

Les heuristiques très simples se sont révélées très efficaces pour réduire significativement le temps de calcul et permettent en pratique de proposer un plus court chemin à travers une agglomération de plusieurs millions d'habitants.

Nous avons cependant essayé d'améliorer encore plus le temps de calcul. Deux approches ont été testées et seront présentées dans les deux sections suivantes.

5.2 Algorithme de Tsaggouris

Nous avons présenté l'algorithme de Tsaggouris dans le premier chapitre. Cette approche

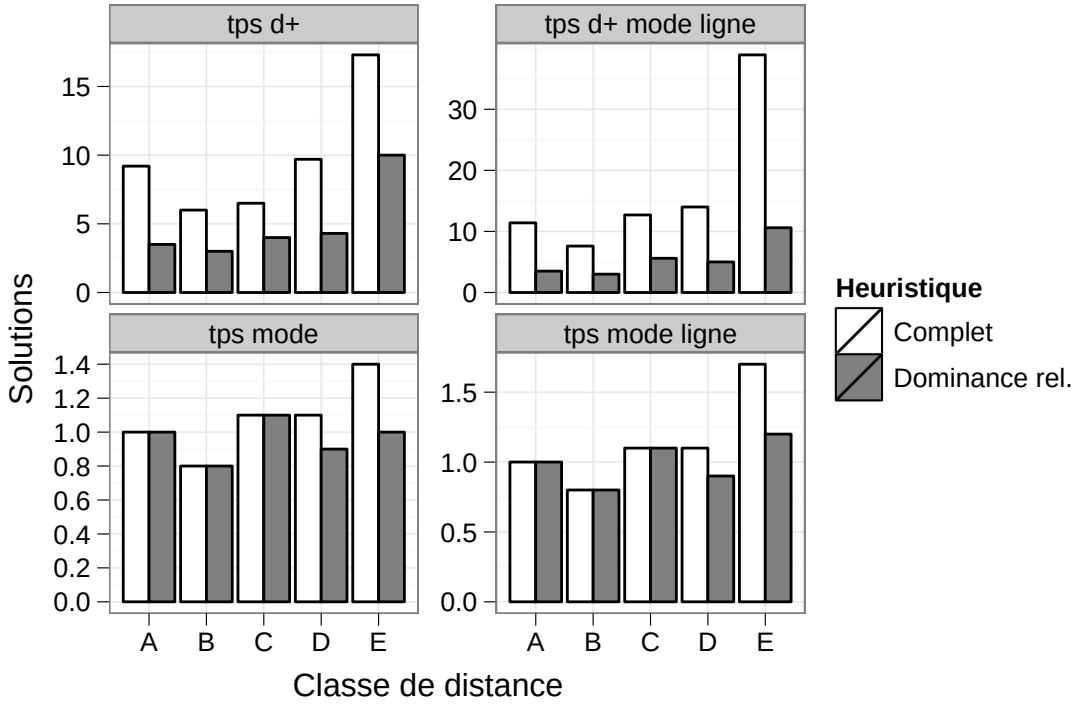


Figure 5.4 Nombre de solutions selon les objectifs et les heuristiques (Los Angeles)

	Nbre nœuds	Nbre arcs	Fréq processeur	Tps de calcul
Gueye (Toulouse)	56 774	146 280	2,47 Ghz	1 600 ms
Gräbener (San Francisco)	14 529	43 788	1,67 Ghz	< 100 ms
Gräbener (Los Angeles)	135 663	375 382	1,67 Ghz	< 800 ms

Tableau 5.1 Comparaison de nos résultats avec ceux de Fallou Gueye (temps et changements de mode)

est très intéressante puisqu'elle permet de calculer le front de Pareto en temps polynomial à un facteur ϵ près.

5.2.1 Limitations

Étant basé sur l'algorithme de Bellman-Ford, il n'est pas possible de garantir l'optimalité des solutions trouvées avant d'avoir effectué les n itérations et donc calculé les plus courts chemins d'un nœud vers *tous* les nœuds.

Malgré cette limitation, nous avons tenu à tester cette approche pour nous faire une meilleure idée des performances.

5.2.2 Expérimentations

Nous avons commencé par faire une implémentation statique afin d'en estimer les performances. En effet, nos expériences du chapitre 2 ont montré que ce qui limite les performances de l'algorithme de Martins est avant tout le nombre d'éléments dans le front de Pareto. Si le nombre d'éléments sur le front est faible, alors il se peut que l'algorithme de Martins soit plus performant car il effectue moins d'itérations.

Pour tous les objectifs, nous avons fixé $\epsilon = 0.1$ et effectué 10 calculs en choisissant le point de départ aléatoirement.

5.2.3 Résultats

Le graphe de test comporte 26 000 nœuds centré sur Paris et les objectifs sont la distance et une mesure du sentiment de sécurité à vélo. Ces données sont issues des travaux de Sauvanet,Néron [58].

Le temps de calcul moyen pour l'algorithme de Martins est de 2.53 secondes tandis qu'il est de 71.1 secondes pour l'algorithme de Tsaggouris.

Les résultats sont particulièrement mauvais. Cela peut s'expliquer par des calculs plutôt lourds (calculer des logarithmes, troncatures) par rapport à l'algorithme de Martins (uniquement des additions). De ce fait, la meilleure complexité algorithmique théorique est largement perdue par le temps nécessaire pour chaque étape élémentaire.

De plus, même si le nombre d'éléments sur le front de Pareto peut augmenter de manière exponentielle, en pratique nous avons vu que ce nombre dépasse rarement la centaine d'éléments (voir la **figure 5.4**). Étant donné que c'est la taille de ce front qui rend l'algorithme difficile, si le front reste de taille modeste, alors il n'existe pas un réel besoin d'avoir un algorithme *théoriquement* performant.

5.2.4 Conclusion

Ces piètres performances ont été décevantes mais auraient pu être prévisibles devant le grand nombre d'opérations nécessaires pour exécuter l'algorithme. Tolérer une marge

d'erreur ne semble pas réduire le front des solutions de manière suffisamment significative pour que l'algorithme de Tsaggouris ne soit plus performant que celui de Martins.

Nous avons donc décidé d'écarter cet algorithme par la suite. Afin de réduire les temps de calcul, nous avons approfondi les algorithmes les plus performants de la recherche d'itinéraire mono-objectifs.

5.3 Contraction hierarchies

Afin d'améliorer les performances nous nous sommes penchés sur les approches haute performance afin de voir si elles pouvaient être adaptées au cas multiobjectif.

Parmi les trois grandes approches que nous avions présentées dans la **sous-section 1.2.3.4**, nous avons éliminé le marquage d'arcs, ainsi que toutes les approches nécessitant d'effectuer un calcul d'itinéaire lors d'une phase de prétraitement. En effet du fait de la topologie changeante d'un réseau de transport en commun entre jour et nuit ainsi qu'entre la semaine et le week-end, mais aussi du fait de l'optimisation multiobjectif, il existe un très grand nombre d'itinéraires possibles. Adapter ces méthodes serait donc particulièrement délicat.

Nous nous sommes intéressés à l'approche *contraction hierarchies* de Geisberger, [5]. Il s'agit en effet d'une des approches les plus simples et des plus performantes à la fois. Elle présente le grand avantage de ne pas calculer à l'avance des itinéraires (même si cela peut être effectué à des fins d'optimisation, cela n'entrave pas son fonctionnement). De plus, les *contractions hierarchies* avaient déjà été adaptées au cas dépendant du temps avec succès Batz, [17].

5.3.1 Adaptation

Comme pour tester l'algorithme de Tsaggouris, nous avons dans un premier temps testé notre adaptation sur un réseau statique afin de tester la facilité d'adaptation.

Ajout des arcs de raccourci

Étant donné les nœuds a, u, b avec un numéro d'ordre arbitraire tels que les arc $((a, u)$ et $(u, b))$ existent et $a > u$ et $b > u$, un arc (a, b) de raccourci est ajouté au graphe. Le coût de ce nouvel arc est la somme des coûts des deux arcs $\mathbf{c}_{ab} = \mathbf{c}_{au} + \mathbf{c}_{ub}$. Si un arc (a, b) existait déjà, le coût de l'arc original et du nouvel arc sont comparés. L'arc ayant un coût dominé est

supprimé. Si les deux arcs ont des coûts non comparables, alors le graphe comportera des arcs parallèles. C'est là que se situe la prin[20]cipale différence avec l'algorithme proposé initialement. Cette étape de contournement du nœud u est appelé *suppression*.

Afin de *contracter* tout le graphe, les nœuds sont *supprimés* selon un ordre arbitraire. Il est possible d'arrêter la contraction à tout moment. Les nœuds qui n'ont pas été supprimés forment ce que la littérature appelle le *core*. En effet, le nombre de raccourcis rajoutés à chaque suppression a tendance à augmenter puisque les arcs de raccourcis peuvent à leur tour être intégrés dans un autre raccourci. Le nombre de raccourcis peut donc devenir très important vers la fin de la contraction.

Ordre des nœuds

Le processus de contraction fonctionne avec n'importe quel ordre. Cependant les auteurs montrent différentes manières de déterminer l'ordre, soit pour accélérer le processus de contraction, soit pour accélérer le calcul d'itinéraire. Nous avons utilisé comme unique critère la différence d'arcs car, d'après la littérature Geisberger, [5], il s'agit de loin du critère le plus efficace. Il s'agit de la différence entre le nombre d'arcs rajoutés et le nombre d'arcs adjacents au nœud. Plus cette différence est petite, plus le nœud sera traité tôt.

Calcul d'itinéraire

Le calcul d'itinéraire est bidirectionnel. La création d'étiquettes se fait comme dans l'algorithme de Martins sauf qu'au lieu de considérer tous les successeurs, seuls les arcs du type (u, v) avec $u < v$ sont considérés.

Deux ensembles d'étiquettes temporaires sont utilisés, un pour chaque direction. Lorsqu'une étiquette est créée pour un nœud ayant déjà été visité dans l'autre sens, alors plusieurs itinéraires complets sont obtenus. Un test de dominance élimine alors les chemins dominés.

Optimalité

Il peut paraître surprenant que le fait de n'explorer que les arcs allant vers des nœuds supérieurs permette d'avoir les mêmes chemins qu'avec l'algorithme de Martins traditionnel. Nous montrons l'équivalence des deux algorithmes : en ayant un chemin obtenu par un algorithme, on peut obtenir un chemin de même coût avec l'autre algorithme.

[20] `http://tinyurl.com/32ntl9u`

Étant donné un chemin obtenu avec l'approche bidirectionnelle présentée précédemment, il est possible de reconstruire le plus court chemin dans le graphe original. Il suffit en effet de « dérouler » chaque arc de raccourci pour obtenir les arcs correspondants au graphe original. Le coût des chemins sera le même puisque le coût des raccourcis est égal à la somme des coûts des arcs court-circuités.

Soit un chemin obtenu par l'algorithme de Martins traditionnel de la forme $u_1, u_2, \ldots, u_k$, soit M le plus grand nœud de ce chemin selon l'ordre défini pour la contraction, chaque triplet de nœuds consécutifs a, u, b tel que $a > u$ et $b > u$ situé entre le nœud de départ et M est remplacé par l'arc de raccourci (a, b) qui existe puisque c'est ainsi qu'a été défini la contraction. En appliquant cette procédure itérativement nous obtenons un itinéraire du nœud de départ à M où tous les nœuds sont dans l'ordre croissant. Cela correspond à l'itinéraire trouvé par la première recherche. La même procédure est appliquée entre M et le nœud de destination afin de n'avoir que des arcs décroissants. Ce segment correspond à la recherche inverse.

Il y a donc une équivalence entre les itinéraires trouvés avec chacun des deux algorithmes.

5.3.2 Expériences

Instance de test

Nous avons comparé notre version multiobjectif des *contraction hierarchies* avec notre implémentation de l'algorithme de Martins.

Le graphe est le même que celui utilisé pour tester l'algorithme de Tsaggouris : centré sur Paris avec 26 000 nœuds et les objectifs sont la distance et une mesure de sentiment de sécurité à vélo.

Nous avons effectué 100 calculs aléatoires en comparant systématiquement les résultats obtenus pour s'assurer de l'absence de bogues dans l'implémentation, ainsi que le nombre d'étiquettes générées. En effet ce nombre permet d'avoir une estimation du nombre d'opérations effectuées. Une autre mesure moins habituelle pour estimer la performance d'un algorithme est le nombre de comparaisons (ou tests de domination) qui représentent une consommation très importante du temps de calcul dans l'optimisation multiobjectif (environ 90% du temps est passé à effectuer des tests de domination sur nos implémentations). Le temps indiqué est le temps moyen sur les 100 calculs.

Résultats

Le nombre moyen d'itinéraires trouvés est 36,8. Le temps de calcul moyen , le nombre moyen d'étiquettes générées ainsi que le nombre de comparaisons en moyenne sont présentés dans le **tableau 5.2**.

Algorithme	Temps moyen	Étiquettes générées	Comparaisons
Martins	821 ms	284 309	10 534 897
Contraction hierarchies	1 293 ms	16 608	14 427 573

Tableau 5.2 Performances des contractions hierarchies multiobjectif

On constate que les temps de calculs sont du même ordre de grandeur dans les deux cas, même si l'algorithme de Martins est légèrement plus rapide. En ce qui concerne le nombre d'étiquettes créées, l'approche basée sur les contractions est nettement plus performante puisqu'il existe en moyenne un facteur 17.

On constate que le nombre de tests de domination semble fortement corrélé au temps de calcul. L'approche contraction hierarchies nécessite un nombre de tests bien plus important à cause de la recherche bidirectionnelle afin de combiner les deux bouts de chemins et tester la domination. Ainsi si les deux recherches trouvent chacune n chemins pour arriver au nœud, n^2 chemins seront générés, ce qui nécessitera $(n^2)^2$ comparaison pour savoir quels sont les chemins dominés. Pour $n = 5$, il y aura donc 625 tests de domination.

5.3.3 Conclusion sur les contractions hierarchies

Les résultats sont étonnants dans la mesure où nous devons étudier significativement moins d'étiquettes avec cette approche qu'avec l'algorithme de Martins. Cependant en temps, les deux approches sont très comparables. Les bonnes performances en termes d'étiquettes sont annulées par un nombre de tests de domination plus important.

Nous pensons que cette approche mérite d'être étudiée de manière plus approfondies afin, nous l'espérons, de pouvoir effectuer moins de comparaisons et donc d'avoir de meilleurs temps de calcul.

Conclusion

Nous avons montré que la simple extension de l'algorithme de Dijkstra avec de simples heuristiques permet d'obtenir des résultats très satisfaisants. Nous avons été déçus par les

performances de l'algorithme de Tsaggouris. En ce qui concerne les *contraction hierarchies*, les performances en terme de nœuds visités sont excellentes alors qu'en terme de temps de calcul elles sont du même ordre qu'en utilisant l'algorithme de Martins. Cette voie est donc un peu décevante, mais pourrait se révéler plus performante si le problème du nombre de comparaisons pouvait être résolu.

Comme nous l'avions supposé dans le chapitre 2, lorsque les objectifs sont très variables (émissions de CO_2 ou dénivelé positif) alors le front de Pareto généré est bien plus grand qu'avec des objectifs souvent nuls (nombre de changements), rendant le calcul bien plus long.

Quand au temps de calcul il reste de l'ordre de la seconde sur les plus grandes instances correspondant donc aux objectifs visés.

Dans certains cas, il est possible qu'un nombre bien trop grand de solutions soient trouvées. Ces solutions étant trop semblables, il n'est souvent pas pertinent de toutes les présenter. Le chapitre suivant propose différentes approches afin de réduire le nombre de solutions proposées.

Exemple de chemins non comparables

Les figures suivantes montrent quatre alternatives de routes entre les mêmes points de départ et d'arrivée. Trois modes sont considérés : piéton, le réseau Muni (bus et tramway) et le réseau Bart (trains péri-urbains). Les objectifs sont le temps, le nombre de changement de ligne à l'intérieur d'un même réseau et les changements de mode (de piéton vers les transports en commun ou d'un réseau de transport en commun à l'autre).

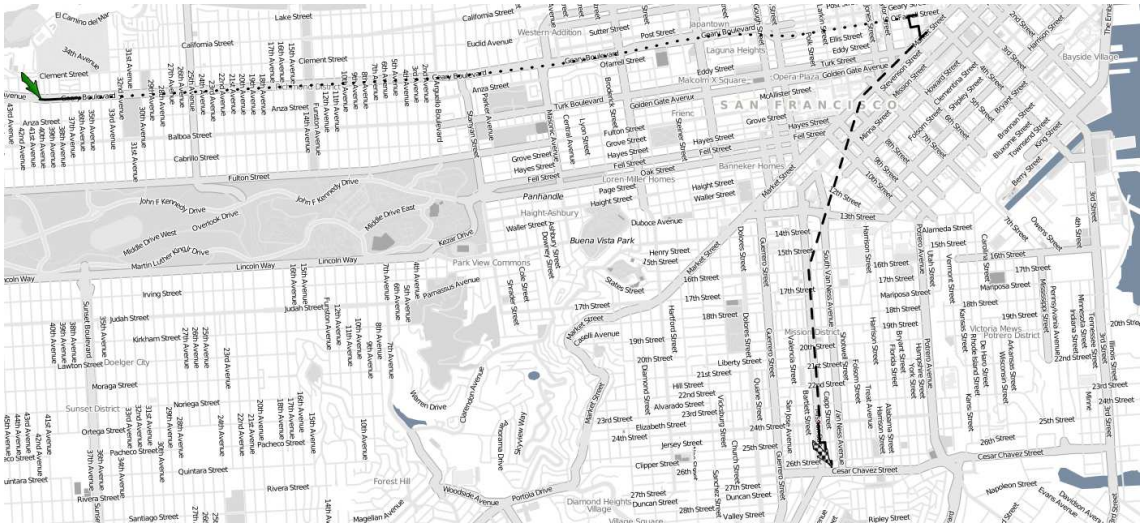


Figure 5.5 Exemple de chemin à San Francisco (54 min, 2 changements de mode, 0 changement de ligne)

Figure 5.6 Exemple de chemin à San Francisco (67 min, 1 changement de mode, 1 changement de ligne)

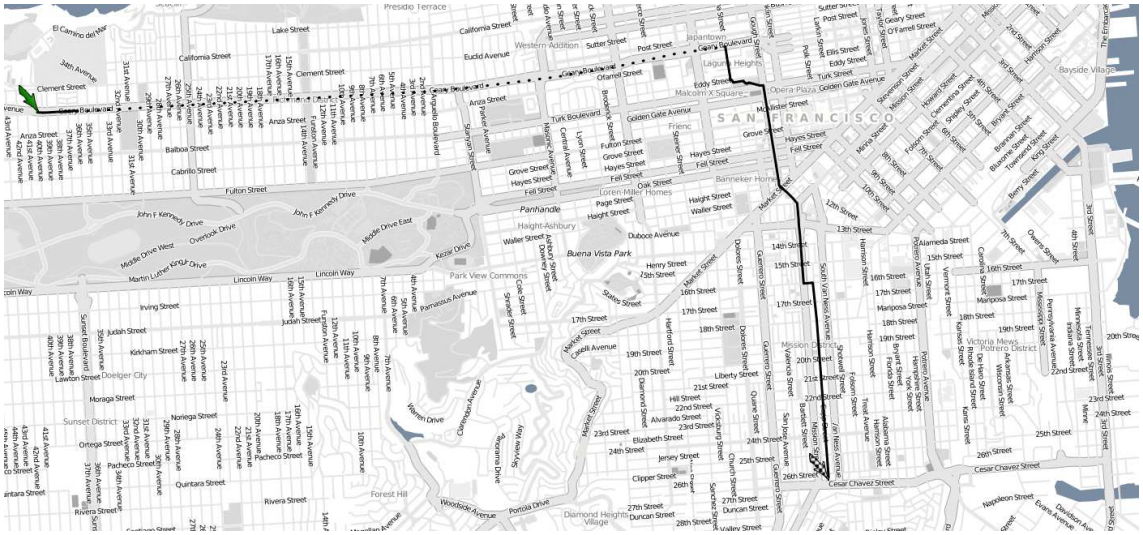


Figure 5.7 Exemple de chemin à San Francisco (83 min, 1 changement de mode, 0 changement de ligne)

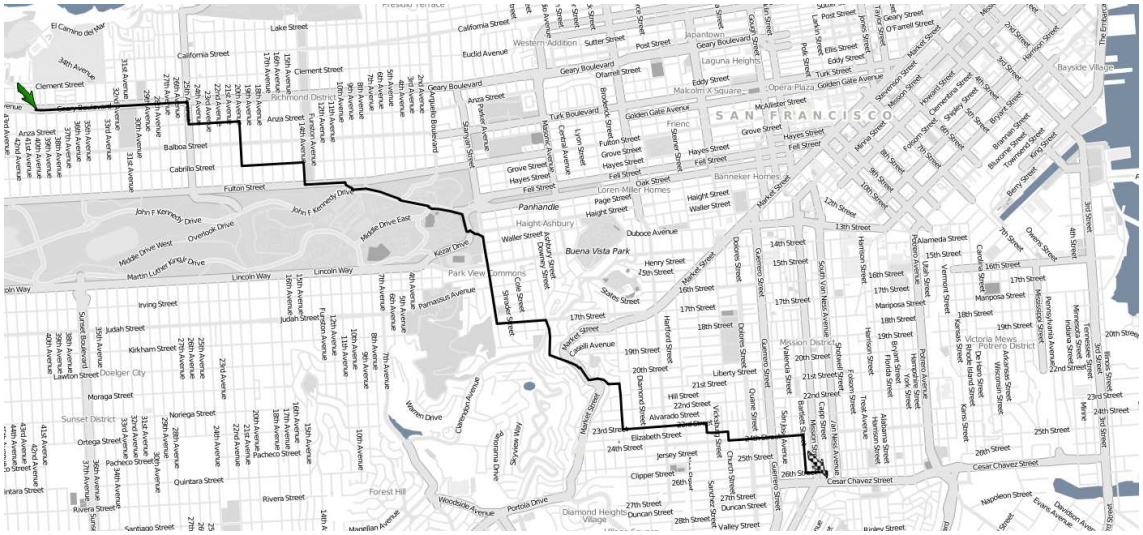


Figure 5.8 Exemple de chemin à San Francisco (83 min, 0 changement de mode, 0 changement de ligne)

Chapitre 6
Sélection d'un sous-ensemble de solutions de qualité

Ne quid nimis — Point trop n'en faut

— Publius Terentius Afer, Thérence

Le chapitre précédent a montré qu'il est possible d'obtenir des solutions Pareto optimales pour des calculs d'itinéraires combinant plusieurs modes de transport. Cependant, pour certains objectifs et suivant la configuration des réseaux des villes, il peut y avoir plusieurs dizaines de solutions équivalentes. Or, si avoir plusieurs solutions est un gain significatif pour l'utilisateur final, lui en présenter des dizaines trop similaires va le décourager d'utiliser le calculateur d'itinéraire.

Il est donc indispensable de ne proposer qu'un nombre plus restreint de solutions et donc de ne garder que les plus pertinentes. Cependant, les approches varient également selon la personne qui définit les critères pour restreindre les choix. Un *expert* agira en amont en préjugeant des préférences de l'utilisateur. Il pourra même en imposer certaines telles que l'obligation de laisser la voiture dans un parking relais plutôt que de donner la possibilité de la laisser à n'importe quel endroit de la ville afin de favoriser les transports en commun. En ce qui concerne le *décideur* (ou utilisateur du système), nous souhaitons qu'il n'ait que très peu de paramètres à régler pour ne pas le décourager d'utiliser le système.

Nous proposons dans ce chapitre trois approches afin de réduire le nombre de solutions trouvées. Elles se différencient par le moment où elles s'appliquent par rapport à la résolution du problème : au cours de la modélisation, au cours de la résolution ou encore *a posteriori*.

6.1 Action lors de la modélisation

Lors de la modélisation des différentes couches des modes de transport, il faut également définir quelles sont les transitions qui sont considérées comme acceptables.

Par exemple, il est tentant de mettre une transition de la couche vélo vers la couche piéton à tous les nœuds pour modéliser le fait qu'il est possible de laisser son vélo à tout endroit de la ville. Cependant, dans le cas d'une longue côte, le système proposera de laisser le vélo à tous les nœuds de la côte afin de réduire le dénivelé positif.

Or, en pratique, un utilisateur ne change de mode de transport que si cela lui permet de gagner du temps, ou s'il s'agit d'un autre mode que la marche (pour prendre le métro par exemple).

De ce fait, en sélectionnant bien lors de la modélisation les nœuds où il est possible de changer de mode de transport, moins de solutions seront proposées à l'utilisateur, mais celles-ci seront probablement plus en adéquation avec ses souhaits.

6.1.1 Expérimentation

Afin de mettre en évidence ces gains, nous avons considéré le graphe de San Francisco avec deux objectifs (le temps et le dénivelé positif) et deux modes de transport (marche et vélo). L'utilisateur part à vélo, mais arrive à pied.

Nous faisons varier le nombre de nœuds où l'utilisateur peut laisser son vélo. Soit à tous les nœuds, soit un nœud sur 10, un nœud sur 100 et enfin un nœuds sur 1 000. Ces nœuds sont tirés aléatoirement.

De plus nous gardons le même découpage en classes de distance que dans les chapitres 4 et 5.

6.1.2 Résultats

Les résultats sont présentés dans la **figure 6.1**. On constate très clairement que moins il y a de transitions, moins il y a de solutions. Les effets de la modélisation sont d'autant plus nets pour de grandes distances. Ainsi le nombre de solutions trouvées peut être divisé par deux dans le meilleur des cas (classe 6 à 8 km).

Le lecteur attentif sera peut-être étonné que nous trouvions plus de solutions équivalentes avec deux modes (vélo personnel et marche), qu'avec les trois modes considérés dans le chapitre précédent (vélo personnel, marche et transports en commun). Cela s'explique par le fait que prendre les transports en commun domine généralement le vélo en temps et dénivelé positif. Dans le cas présent, la marche est toujours meilleure que le vélo en terme de dénivelé positif et toujours plus mauvaise en temps de trajet.

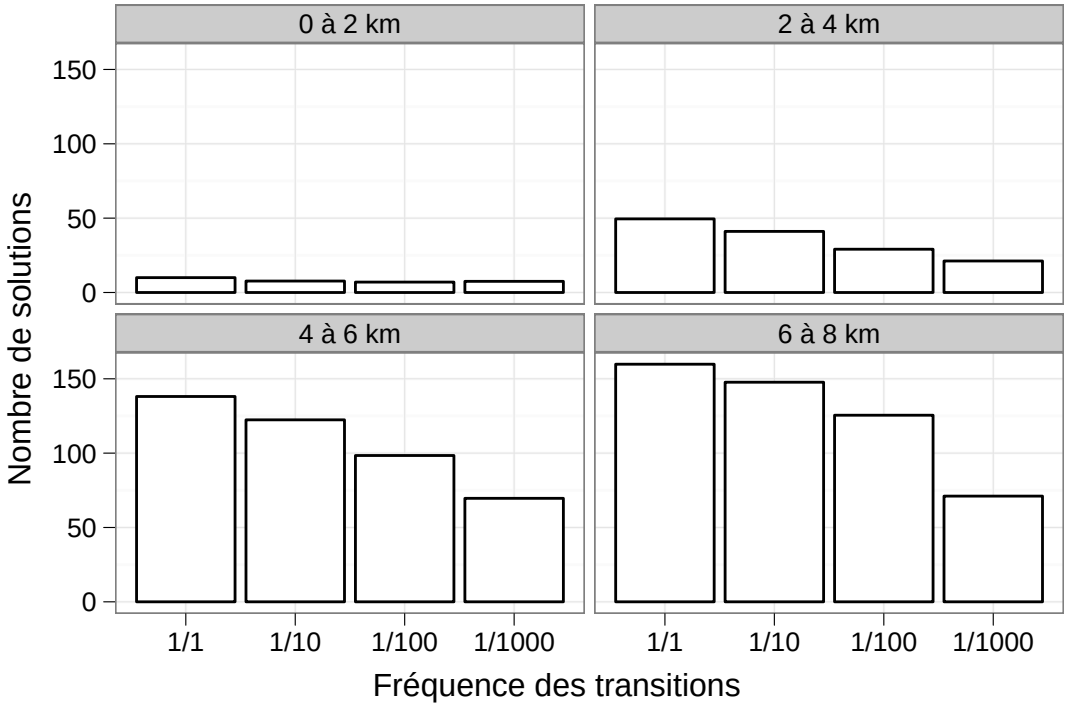


Figure 6.1 Nombre de solution en fonction de la distance et de la fréquence de transition entre modes

6.1.3 Conclusion

Les gains possibles par cette approche sont limités, mais elle présente cependant l'avantage de permettre à l'expert de définir les transitions qu'il considère comme intéressantes. Symétriquement, cela pourrait exclure une possibilité qu'aurait préféré l'utilisateur.

Cependant, proposer 70 itinéraires à un utilisateur alors que seulement deux modes et deux objectifs sont considérés n'est pas intéressant. Il est donc nécessaire de s'intéresser à d'autres approches.

6.2 Action pendant la recherche d'itinéraire : dominance relâchée

Afin d'améliorer les performances de l'algorithme de Martins, nous avons utilisée la dominance relâchée. Cette approche consiste à considérer qu'une première solution domine une autre même si elles sont équivalentes selon une mesure totalement arbitraire. Par

exemple un itinéraire avec 3 changements et 25 minutes de trajets est dominé par un itinéraire comportant 2 changements et 27 minutes de trajet. En effet on peut considérer qu'effectuer un changement de plus pour uniquement gagner deux minutes n'est pas intéressant.

La dominance relâchée a été particulièrement efficace pour améliorer significativement les performances. Elle permet également de réduire au cours de la résolution le nombre de solutions trouvées.

6.2.1 Expérimentation et résultats

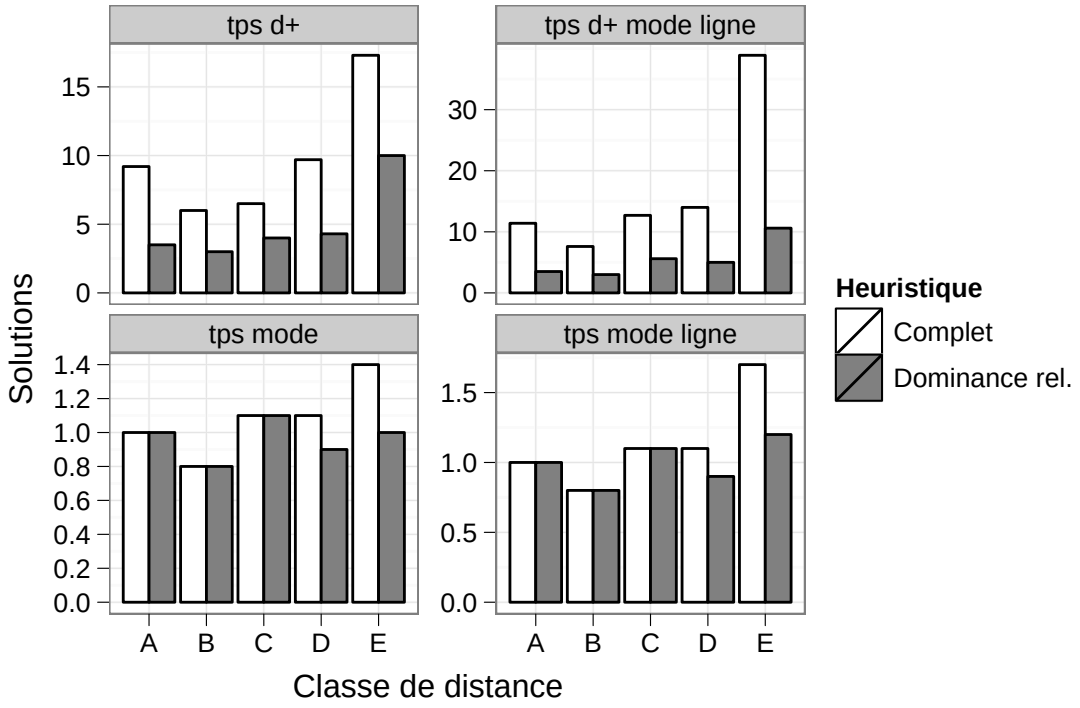


Figure 6.2 Nombre de solutions selon les objectifs et les heuristiques (Los Angeles)

Nous présentons dans la **figure 6.2** les mêmes résultats que dans le chapitre précédent appliqués à la ville de Los Angeles pour démontrer l'efficacité de la dominance relâchée pour réduire le nombre de solutions trouvées.

On constate ainsi que lorsqu'il existe de nombreuses solutions, alors elles peuvent être divisées par trois (avec quatre objectifs). Par contre lorsque peu de solutions existent, la réduction est moins flagrante.

6.2.2 Application

Cette approche peut être appliquée au niveau du décideur qui choisira les paramètres qui permettent de trancher si un itinéraire est dominé, ou alors au niveau de l'utilisateur qui indique le temps qu'il serait prêt à sacrifier pour économiser un changement, un kilo de CO_2, etc. Il est cependant important de se méfier de trop vouloir limiter les solutions au risque de se retrouver avec trop peu de solutions pour qu'une approche multiobjectif en vaille la peine. L'intérêt d'une approche multiobjectif vise à s'éloigner justement d'un jugement précoce de ce que l'on considère comme acceptable.

De plus une contrainte d'application assez forte est qu'entre deux solutions, il est nécessaire de toujours garder celle qui arrive le plus tôt pour des contraintes d'optimalité.

6.2.3 Conclusion

La dominance relâchée est une approche tentante pour sa simplicité de modélisation de cas considérés comme aberrants par l'expert ou l'utilisateur. De plus elle permet de significativement réduire le temps de calcul.

Cependant, il peut malgré tout rester plusieurs dizaines de solutions. Il est alors possible de relâcher encore plus la dominance et risquer de ne plus avoir qu'une seule solution pour des calculs plus simples.

Il serait donc intéressant de pouvoir sélectionner un nombre arbitraire de solutions quelle que soit la taille du front de Pareto.

6.3 Action *a posteriori* : classification des solutions

Nous souhaitons ne retenir que quelques solutions représentatives de l'ensemble du front de Pareto. Nous nous sommes donc intéressés aux outils statistiques de classification. L'idée est de regrouper les solutions dans k classes de résultats et de ne présenter qu'une seule solution de chaque classe.

Nous avons donc appliqué une technique de classification bien connue, les *k-means* (ou *k-moyennes*) MacQueen, [78]. En définissant à l'avance le nombre de classes souhaitées, cette approche regroupe chaque élément dans une classe de manière à minimiser la distance quadratique à la solution moyenne de la classe. Nous avons choisi cette approche parce qu'elle permet de définir à l'avance le nombre de classes que nous souhaitons. De plus elle est simple à mettre en œuvre et nécessite peu de calculs.

6.3.1 Procédure

Étant donné l'ensemble du front de Pareto, les objectifs sont normalisés et l'algorithme des k-means est appliqué. La normalisation est nécessaire pour que la mesure de distance ait un sens. En effet, si le nombre de changements ne dépasse pas la demi-douzaine, le temps de trajet peut facilement dépasser mille secondes. De ce fait le nombre de changements serait considéré comme négligeable par rapport au temps de parcours malgré l'importance de ce paramètre.

Pour chaque classe on sélectionne la solution la plus proche du barycentre.

Cette approche étant très classique, nous avons utilisé une implémentation existante. Le temps de calcul avec une centaine d'éléments est négligeable par rapport au temps de calcul de l'itinéraire à proprement parler et n'est pas pénalisant.

6.3.2 Exemple

Afin d'illustrer cette approche, nous avons choisi un itinéraire comportant de nombreuses solutions équivalentes. Les objectifs sont le temps, le dénivelé positif et le nombre de changements de modes. Les modes de transports pris en compte sont le vélo, les transports en commun et la marche.

Les figures **6.3** et **6.4** montrent les fronts de Pareto et la classification en respectivement trois et cinq catégories. Chaque courbe correspond au front de Pareto obtenu en ne gardant que les solutions avec 1, 2 et 3 changements de mode.

6.3.3 Nécessité de la normalisation

La **figure 6.5** représente les solutions obtenues avec trois catégories sans avoir appliqué la normalisation. Par rapport au résultat présenté dans la **figure 6.3**, on constate que la classification n'est pas la même. En effet, sans la normalisation le nombre de changements est considéré comme un critère ne présentant que peu de variation par rapport au temps de parcours et au dénivelé alors que cela a une très grande importance pour l'utilisateur. C'est pour cela que l'on constate que trois solutions avec deux changements sont dans la même catégorie que s'il n'y a qu'un seul changement dans la **figure 6.5**.

6.3.4 Bilan des *k-means*

L'utilisation des *k-means* nous permet de trouver un solution représentative pour chaque classe. Le nombre de classes peut être librement défini par l'utilisateur. Il s'agit aussi

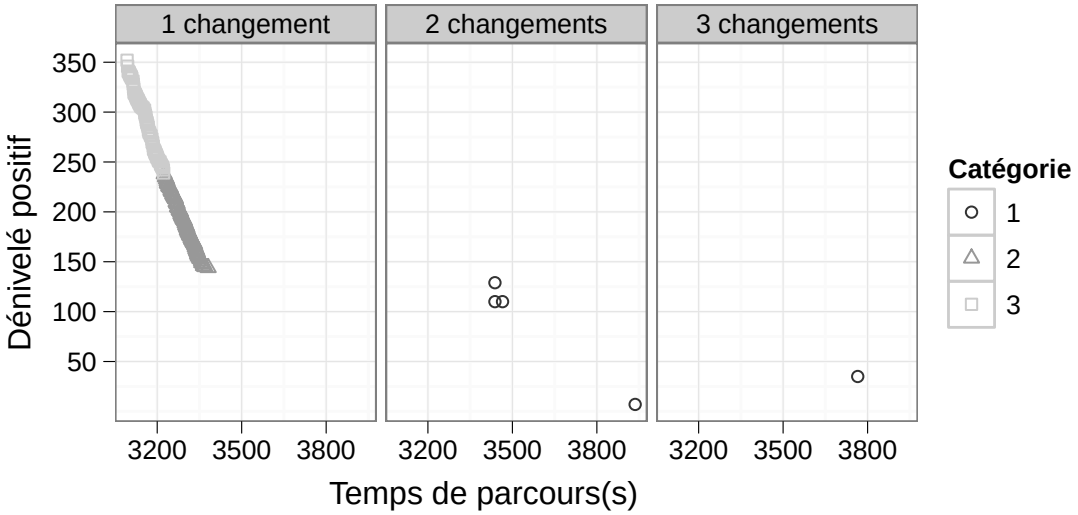


Figure 6.3 Classification des résultats, 3 catégories, avec normalisation

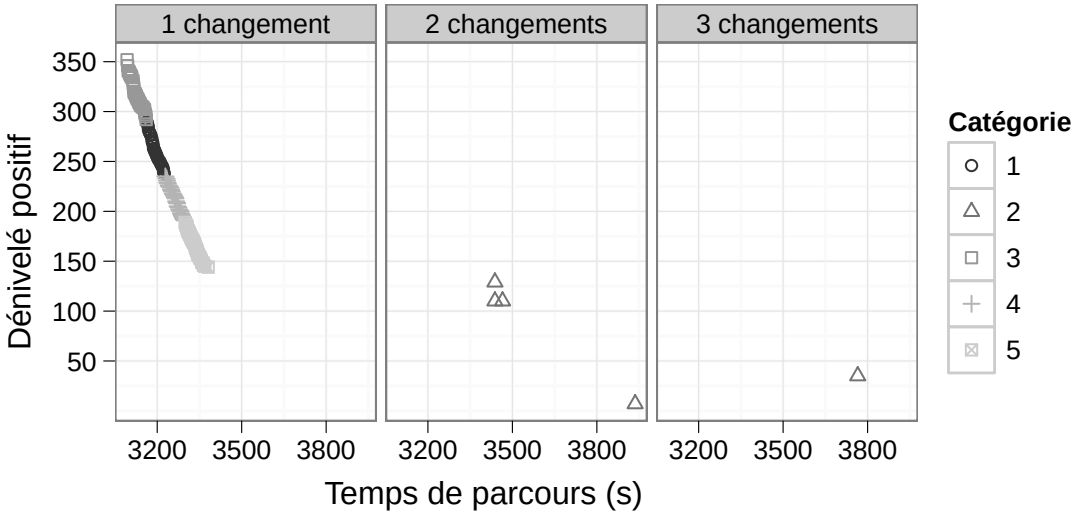


Figure 6.4 Classification des résultats, 5 catégories, avec normalisation

bien d'un avantage que d'un défaut. En effet, il se peut que dans certains cas, il existe un nombre de classes qui serait plus naturel qu'un nombre arbitraire. Pour traiter ce genre de cas, il pourrait être intéressant d'étudier également des algorithmes de partitionnement qui ne nécessitent pas de définir à l'avance un nombre de classes.

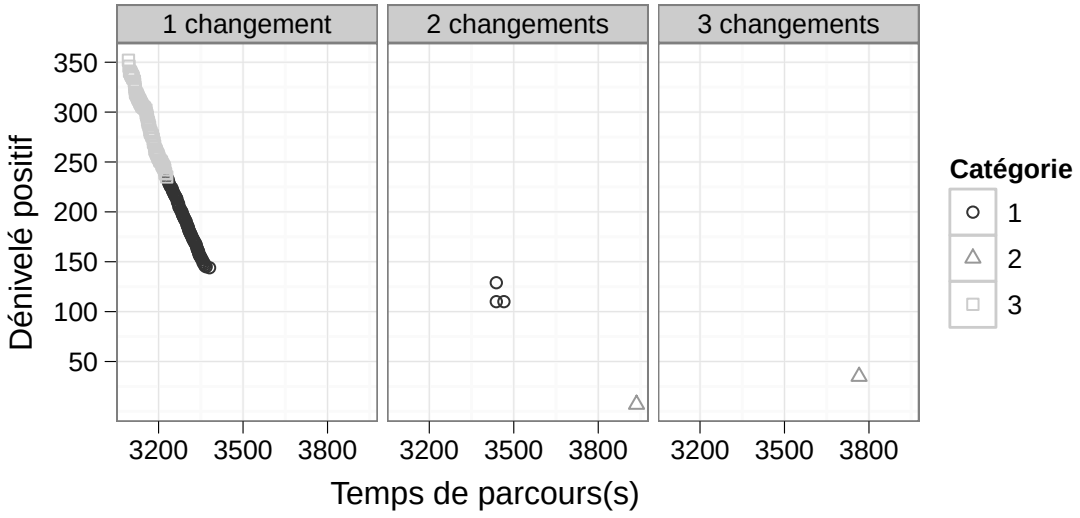


Figure 6.5 Classification des résultats, 3 catégories sans normalisation

Conclusion

Nous avons présenté trois approches pour réduire le nombre de solutions à présenter à l'utilisateur. Ces approches sont complémentaires et permettent aussi bien d'intervenir en amont du calcul d'itinéraire qu'en aval. Deux des trois approches peuvent également être paramétrées par l'utilisateur qui souhaiterait avoir une gestion plus fine des paramètres retournées.

Bien évidemment les paramétrages et les approches utilisées dépendront de l'utilisation souhaitée.

À ce point nous avons posé les bases pour construire un calculateur d'itinéraire multimodal et multiobjectif depuis la modélisation jusqu'au traitement des données. Le prochain chapitre présente quelques variantes pouvant être appliquées directement à notre modèle et les algorithmes proposés.

Chapitre 7
Variantes simples

'Would you tell me, please, which way I ought to go from here ?'
'That depends a good deal on where you want to get to,' said the Cat.
'I don't much care where —' said Alice.
'Then it doesn't matter which way you go,' said the Cat

— Lewis Carroll, Alice's Adventures in Wonderland

Comme dans toute proposition de modèles et d'algorithmes, le lecteur sera probablement intéressé par les extensions possibles pour des applications similaires. Malheureusement — ou heureusement — il y aura toujours plus d'idées et il n'est donc pas possible de toutes les mettre en œuvre.

Ce chapitre présente des détails de l'algorithme que nous avons sciemment laissé de côté par simplicité. Nous proposons également quelques adaptations qui nous semblent intéressantes et qui ne devraient pas poser de problème particulier de mise en œuvre.

7.1 Optimiser le temps

Depuis le début nous avons considéré que l'utilisateur part à un instant t_0. De ce fait, l'utilisateur peut se retrouver à attendre 20 minutes à un arrêt de bus (par la suite le nœud u). S'il était parti 20 minutes plus tard, il serait arrivé exactement au même moment.

Pour corriger cela, il suffit pour chaque itinéraire de remonter le temps (voir la section d'après) à partir du nœud u. On obtient ainsi l'heure de départ la plus tardive possible.

Il n'est pas possible de simplement soustraire le temps d'attente au nœud u puisque le temps de trajet peut varier selon l'instant où l'arc est parcouru (par exemple en se retrouvant dans des embouteillages).

Cette approche reste malgré tout optimale, car la contrainte FIFO nous garantit que partir plus tard nous fera forcément arriver plus tard. De plus les coûts sont également croissants avec le temps (sauf en un nombre finis de points à traiter au cas par cas).

7.2 Heure d'arrivée précise

Le système que nous avons proposé nécessite la définition d'une heure de départ t_0 par l'utilisateur. Or celui-ci peut vouloir fixer une heure d'arrivée (par exemple pour arriver à l'heure à un rendez-vous).

7.2.1 Inversion du graphe

L'adaptation consiste à inverser le sens des arcs. Tout arc (u, v) sera remplacé par un arc (v, u) ayant le même coût. Pour les arcs associés à des transports en commun qui suivent une grille horaire, la transformation est moins triviale. En effet, il est impossible d'utiliser des fonctions de coût décroissantes. Il est donc nécessaire de modifier la manière dont est calculé le temps.

7.2.2 Modification du calcul du temps

Dans les chapitres précédents, par convention la première valeur c_0 du vecteur de coûts **c** indique un instant. Nous modifions cela en mémorisant le temps de parcours depuis le début. L'instant est donc désormais $t_0 - c_0$, soit l'heure d'arrivée souhaitée moins le temps pour arriver au nœud.

Les fonctions de coûts doivent donc retourner le temps de parcours et le temps d'attente à l'arrivée pour arriver à l'instant $t_0 - c_0$. Dans le cas de coûts constants, il s'agit tout simplement de la durée de parcours. Dans le cas d'un horaire, il faut rechercher dans le tableau d'horaires le dernier bus ou train permettant d'être présent à $t_0 - c_0$.

Soit un trajet en train du nœud u au nœud v avec un train partant à 8h et à 8h15. Le temps de trajet est de 30min. Le trajet du nœud v à la destination t se fait à pied et nécessite 20 minutes. On souhaite arriver à t à 9h00. On a donc $t_0 = t_t =$ 9h00 et $c_0(t) = 0$. En v on a $c_0(v) =$ 20min et donc $t_v = t_t - c_0(v) =$ 8h40. Il sera donc nécessaire de prendre le train de 8h pour arriver à t à 9h00.

7.2.3 Implémentation

D'un point de vue implémentation, il suffit de redéfinir les fonctions de mise à jour de c_0 lors de la création d'une nouvelle étiquette et obtenir le temps de parcours dans le cadre d'horaires.

En pratique il n'y a pas besoin de modifier la structure de données en elle-même, car la structure de graphe de Boost permet de parcourir tous les arcs dans le sens inverse.

7.3 Départs décalés

Nous proposons à l'utilisateur plusieurs solutions équivalentes étant donné une heure de départ fixe. Si l'utilisateur a des horaires plus souples, il peut souhaiter avoir plusieurs propositions décalées dans le temps (comme cela est le cas sur la majorité des calculateurs existants).

Le problème réside dans l'impossibilité de savoir au nœud de départ les différents modes de transport utilisant une grille horaire pendant un certain intervalle.

7.3.1 Lancer plusieurs calculs d'itinéraires

L'approche la plus immédiate est de faire plusieurs calculs en décalant le temps initial à chaque fois de 10 minutes par exemple.

Avec l'optimisation du temps de départ, cette approche risque de renvoyer deux fois le même itinéraire. Il est donc nécessaire d'effectuer un tri dans les solutions retournées pour éviter ce cas. De plus avec cette approche il sera possible de proposer deux itinéraires dont l'un domine fortement l'autre, mais dont l'instant de départ est décalé et peut donc malgré tout présenter un intérêt à l'utilisateur.

7.3.2 Utiliser plus d'étiquettes de départ

Une autre possibilité est de peupler la liste des étiquettes temporaires du nœud de départ avec plusieurs étiquettes correspondant à des décalages dans le temps. Il est alors indispensable lors des tests de domination de comparer le temps parcouru et non pas l'instant d'arrivée.

La contrepartie de cette approche est que si les itinéraires sont les mêmes (par exemple avec un bus partant toutes les 10 minutes tous les objectifs étant constants), alors les tests de domination élimineront toutes les solutions à l'exception d'une seule.

7.3.3 Créer plusieurs étiquettes pour les grilles horaires

Une approche plus compliquée à mettre en œuvre consisterait à s'intéresser à plusieurs départs possibles depuis nœud et donc générer plusieurs étiquettes étant donné un nœud

et un successeur. Cependant, cela ne doit être effectué que pour le premier arc régi par une grille horaire. De ce fait, l'implémentation deviendra plus compliquée.

7.4 Départs et arrivées multiples

L'utilisateur peut vouloir considérer plusieurs points de départ et d'arrivée (ou différents modes pour comparer si partir à pied, à vélo ou en voiture est plus intéressant). Pour cela il suffit de créer une étiquette initiale pour chaque nœud de départ souhaité. L'algorithme de Martins étudiera par lui-même les différentes possibilités en calculant les itinéraires depuis tous ces nœuds.

À des fins de meilleures performances, l'heuristique qui consiste à tester la domination par une étiquette du nœud d'arrivée doit être adaptée pour prendre en compte tous les nœuds d'arrivée.

Cette approche ne garde que les itinéraires Pareto optimaux et ne calcule pas tous les itinéraires entre toutes les paires de nœuds.

7.5 Parallélisation

Les temps de calcul que nous nous étions fixés est acceptable pour une personne. Cependant, si le système est mis en place sur un serveur web qui connaît un certain succès il sera nécessaire de réfléchir à la gestion de plusieurs calculs simultanés afin de maintenir un temps de calcul acceptable.

Étant donné de la généralisation des processeurs à plusieurs cœurs, il serait dommage de ne pas pouvoir en profiter. Cependant cela nécessite de mettre en place une parallélisation de l'algorithme.

7.5.1 Paralléliser l'algorithme

Il pourrait être tentant de paralléliser l'algorithme en lui-même. En effet, il serait possible d'extraire plusieurs étiquettes de la file de priorité et de les traiter en parallèle. Le gain ne sera cependant probablement pas linéaire au nombre de cœurs travaillant en parallèle. En effet nous risquons d'explorer les successeurs d'une étiquette qui sera dominée par la suite rendant l'exploration superflue. De plus l'accès aux différentes structures de données (file de priorité et ensembles temporaires) doivent être protégé pour éviter qu'elles

soient modifiées simultanément. Cela induira une plus grande complexité du code, des ralentissements possibles et un risque accru de bogue.

7.5.2 Paralléliser les requêtes

Paralléliser l'algorithme permettrait d'améliorer les performances de l'algorithme au détriment d'une plus grande complexité. Cependant individuellement, les temps de calcul sont acceptables et ce qui nous intéresse est plutôt d'être capable de servir un grand nombre de demandes d'itinéraires simultanément.

L'implémentation sépare clairement deux structures de données : d'un côté le graphe avec les fonctions de coûts qui ne changent pas et de l'autre côté les données spécifiques à un calcul d'itinéraire. Il serait donc à la fois plus simple et plus performant de ne pas modifier l'algorithme mais de se contenter d'exécuter plusieurs recherches en parallèle. L'accès au graphe sera partagé par toutes les recherches, mais comme il s'agit de lecture seule, il n'y aucune nécessité de protéger l'accès aux données.

7.6 Isochrones

La version originale de l'algorithme de Martins calcule des itinéraires d'un nœud vers *tous* les nœuds. De ce fait, en supprimant le test de domination par une étiquette au nœud d'arrivée, on peut construire des isochrones.

On peut ainsi déterminer l'ensemble des points accessibles en moins de 30minutes, moins de 2 changements etc. Une application concrète peut être pour choisir un logement en fonction du lieu de travail.

Conclusion

Nous avons présenté quelques variantes qui peuvent être facilement appliquées. Cela confirme la généricité de notre approche. Celle-ci peut donc être appliquée à de nouveaux besoins.

Sans nul doute, il existe d'autres variantes faciles à mettre en place, mais également des questions plus difficiles et il ne serait pas possible de tout traiter. Nous concluons donc cette thèse en laissons ces questions ouvertes pour qu'elles soient traitées par d'autres personnes.

Conclusion et perspectives

Conclusion

Nous avons souligné en introduction de cette thèse le besoin urgent d'un report de la voiture comme mode de transport vers d'autres modes pour lutter contre les émissions de gaz à effet de serre. Nous pensons qu'une partie de ces changements de comportements pourra se faire grace à une meilleure information sur les interconnexions entre différents modes de transport. Ma contribution au travers de cette thèse a été de présenter un modèle et des algorithmes permettant de présenter à l'utilisateurs plusieurs itinéraires équivalents entre deux points. Ainsi l'utilisateur pourra comparer objectivement les alternatives et envisager d'abandonner la voiture sur certains trajets.

L'état de l'art a permis de montrer la recherche très active dans le domaine des calculs d'itinéraires depuis plusieurs décennies y compris ces dernières années avec de nouvelles approches très haute performance. Cependant aucune approche ne satisfaisait nos besoins de généricité, performance et simplicité pour un calcul d'itinéraire multimodal et multiobjectif.

Une première étude a posé les conditions que doivent remplir les fonctions de coûts afin que des solutions exploitables existent (un nombre dénombrable de chemins de longueur finie). Ensuite nous avons caractérisé expérimentalement le comportement de différents types de coût sur des graphes artificiels et issus de réseaux routiers réels.

Par la suite nous avons proposé un modèle qui peut sembler trivial *a posteriori* mais qui présente des caractéristiques très intéressantes. En effet, il est à la fois très simple à comprendre, n'impose pratiquement aucune restriction sur les déplacements acceptés et permet d'utiliser des algorithmes traditionnels. Ainsi il a été possible d'utiliser une implé-

mentation existante de l'algorithme de Dijkstra pour le calcul d'itinéraire multimodal le plus rapide.

Afin de pouvoir optimiser simultanément plusieurs objectifs, nous avons soit adapté des algorithmes multiobjectifs afin de prendre en compte la dépendance au temps (algorithmes de Martins et de Tsaggouris) ou adapté un algorithme haute performance au cas multiobjectif (*contraction hierarchies*). La comparaison de ces différentes approches a montré que l'algorithme de Martins présente les meilleures performances (en plus d'être l'approche la plus simple). Par rapport aux rares travaux comparables auquels nous avons pu avoir accès, nous obtenons de meilleures performances avec une approche plus simple et plus extensible.

L'application sur des instances réelles a montré qu'entre deux point il pouvait y avoir plus d'une centaine de solutions. Ce nombre de solutions étant bien trop important pour l'utilisateur, nous proposons plusieurs manières de réduire le nombre de solutions présentées à l'utilisateur. La sélection de quelques itinéraires significatifs peut se faire au niveau de la modélisation, en définissant des critères pour considérer que deux chemins sont trop similaires ou encore en utilisant des outils statistiques de classification.

Enfin, nous avons présenté quelques pistes pour implémenter assez simplement plusieurs variantes.

Nous avons donc atteint les objectifs que nous nous étions fixés en début de thèse, à savoir une approche simple, facilement extensible avec des performances de l'ordre de la seconde à l'échelle d'une grande agglomération telle que Los Angeles.

Malgré ces résultats très satisfaisants, la problématique est loin d'être close et de nombreuses questions restent ouvertes.

Perspectives

Améliorer les performances

La perspective la plus immédiate est l'amélioration des performances dans l'optique de passage à l'échelle. En effet, en routage mono-objectif, pour minimiser le temps de parcours, il est possible de calculer un itinéraire à travers la planète entière en environ une milliseconde. Nous sommes encore très loin d'avoir de telles performances. Parmi les pistes à étudier, il y a les recherches bidirectionnelles, les approches hiérarchiques, mais aussi le calcul de bornes pour approcher les itinéraires.

Plus simplement, une amélioration de l'implémentation pourrait également améliorer dans une certaine mesure les temps de calcul.

Tournées

Les tournées au sens large ont de nombreuses applications et méritent d'être étudiées dans une optique multimodale et multiobjectif. Une application immédiate des tournées consiste à proposer une boucle fermée de points à visiter (un touriste désirant visiter différents musées ou une personne devant effectuer différentes tâches réparties sur la ville).

Les transports à la demande et le covoiturage se développent ces dernières années. Cependant ces modes de transport ne sont pas intégrés dans les calculateurs existants et manquent donc de visibilité. La difficulté de ces modes de transport est que les itinéraires dépendent du choix d'une seule personne. Il faut donc trouver une manière de représenter des trajets qui ne peuvent exister que si deux personnes se mettent d'accord.

Nouvelles métriques

Il est peu probable qu'il existe une solution idéale pour ne présenter qu'un nombre restreint de solutions à l'utilisateur. Il s'agit en effet d'une part d'un compromis entre un faible paramétrage par l'utilisateur et laisser l'utilisateur définir au mieux ses souhaits. D'autre part, il s'agit d'un compromis entre la sélection de solutions les plus variées et l'élimination de solutions trop extrêmes.

Cette question mérite donc plus d'attention et de nouvelles métriques pour sélectionner quelques solutions significatives. En particulier, la robustesse d'un chemin est un critère particulièrement important : un chemin un peu plus long pourrait être préféré si le risque de tomber dans un embouteillage est faible. De plus, au lieu de mesurer la distance entre deux solutions selon les objectifs, il serait pertinent de mesurer à quel point les itinéraires sont différents : rien ne sert de présenter deux itinéraires qui ne diffèrent que sur 100m.

Changements dynamiques

Nous avons considéré tout le long de cette thèse que tout le graphe est connu à l'avance et qu'il est fiable. Cependant il est indispensable de pouvoir prendre en compte des perturbations ponctuelles (panne, grève, accident, embouteillage, etc.) et donc de modifier le

graphe de manière à pouvoir à la fois traiter la situation perturbée et la situation normale pour l'utilisateur qui prévoit un trajet à plus long terme.

Le système devrait également être capable de prendre en compte une certaine tolérance pour prendre en compte le risque d'embouteillage. La manière de modéliser l'incertitude est déjà un problème en soit : est-ce qu'il suffit de définir un intervalle, ou faudrait-il s'intéresser à des fonctions statistiques ?

Application au transport de marchandises

Nous nous sommes focalisés exclusivement sur le transport de personnes. Cependant la combinaison de plusieurs modes de transports est également une problématique de plus en plus importante pour le transport de marchandises. En plus des modes de transport traditionnels (camion, avion, bateau, péniche, train), la mise en place du ferroutage et des autoroutes de la mer (qui consistent à mettre les remorques de camion directement sur un train ou un navire pour éviter les transbordements des marchandises) offrira des alternatives de plus en plus attrayantes pour les transporteurs.

Annexe A
Implémentation

Le développement des algorithmes et la création d'un prototype s'est fait dans un projet que nous avons nommé *Mumoro* pour « MUltimodal and MultiOjective ROuting ».

Cette annexe présente quelques détails d'implémentation. Nous présentons dans un premier temps l'architecture générale du projet pour avoir un démonstrateur complet. La deuxième partie s'intéresse à des problèmes plus spécifiques aux algorithmes.

Tout le code source est disponible sous la licence libre GPLv3 et est hébergé à l'adresse suivante :

`http://github.com/Tristramg/mumoro/`

Un site de démonstration appliqué à la ville de Rennes est accessible au moment de la rédaction du document (été 2010) à :

`http://mumoro.openstreetmap.fr/`

A.1 Architecture du projet

A.1.1 Organisation générale

Les algorithmes de calcul d'itinéraire sont écrits en C++. L'interface utilisateur se présente sous la forme d'une page web. Afin de gérer l'interconnexion entre les deux, nous avons utilisé des *bindings* python. Enfin, nous avons dû développer certains outils afin de traiter les données.

A.1.2 Traitement des données

A.1.2.1 Openstreetmap

Le traitement le plus complexe a été pour les données routières issues de Openstreetmap. En effet le projet avait initialement été pensé pour un rendu graphique sans penser aux applications de routage. De ce fait les données ne sont pas disponibles sous la forme d'un graphe traditionnel arcs/nœuds.

Nous avons donc développé un outils spécifique *Osm4routing* également disponible sous licence libre (`http://github.com/Tristramg/osm4routing`) qui permet d'avoir les données dans un format adapté à un routage.

La difficulté vient du fait qu'il est nécessaire d'étudier toutes les lignes afin de détecter les interconnexions qui correspondent à un nœud dans un graphe. Le volume des données à manipuler rend la tache délicate (ainsi la France était représentée en juin 2010 par un fichier XML de 7.5 Go ce qui empêche de le charger entièrement en mémoire même sur un ordinateur haut de gamme).

A.1.2.2 Transports en commun

Suite à une initiative de Google, de nombreux réseaux de transport diffusent leurs données au format GTFS (General Transit Feed Specification) qui est en fait un ensemble de fichiers texte au format CSV (Coma Separated Values). Le traitement de ces données est donc très simple, d'autant plus qu'un outil (*transitfeed*) est disponible pour les manipuler.

A.1.2.3 Vélo en libre service

Les vélos en libre service sont très simples à traiter puisqu'il ne s'agit que d'un tableau contenant les coordonnées, l'identifiant de la station et le nombre de vélos disponibles. Quelque soit le format des données, le traitement est trivial.

Dans le cas de la ville de Rennes, ces données sont au format XML accessibles au travers d'une interface web de type REST (afin d'avoir des données en temps réel sur l'état des stations de vélo).

A.1.3 Interface web

L'interface web se compose d'un côté client en HTML et Javascript et de l'autre d'un serveur web en python qui interagit avec les algorithmes.

La **figure A.1** présente le prototype que nous avons développé avec un exemple d'itinéraire. Sur la gauche il est possible de voir le tableau correspondant à trois exemples d'itinéraire.

A.1.3.1 Outils utilisés côté client

L'interface que voit l'utilisateur final est une page web. Toute la partie cartographie est gérée par la bibliothèque Javascript *OpenLayers* qui se charge d'afficher les fonds de carte, dessiner l'itinéraire et permettre l'interaction avec l'utilisateur (déplacements et zoom sur la carte). Nous avons utilisé jQuery afin d'améliorer sensiblement le confort d'utilisation, mais surtout pour gérer les échanges de données avec le serveur.

A.1.3.2 Outils utilisés côté serveur

Le serveur web que nous avons utilisé est *CherryPy* pour sa simplicité de déploiement et pour faciliter l'intégration des algorithmes grâce aux *bindings* python.

A.1.3.3 Interactions

L'utilisateur doit juste définir le point de départ et d'arrivée. Pour cela il peut soit cliquer sur la carte, soit entrer une adresse. Lorsque le départ et l'arrivée sont clairement définis, le client web envoie une requète en AJAX au serveur. Le serveur exécute alors le calcul d'itinéraire, met en forme les données brutes (l'algorithme ne retournant qu'une liste de nœuds pour représenter un chemin) et les renvoie au client.

A.1.3.4 Algorithme de plus court chemin

Le cœur du projet est développé intégralement en C++. Nous avons utilisé différents composants de la bibliothèque *Boost*. En particulier la Boost Graph Library (BGL) nous offre une structure de graphe très facile à paramétrer pour modéliser plusieurs coûts dont certains dépendent du temps. Cependant les algorithmes multiobjectifs ont été développés par nous-mêmes.

A.2 Détail des algorithmes

Nous précisons dans cette section quelques points très spécifiques d'implémentation.

A.2.1 Gestion de la file de priorité

Dans l'algorithme de Martins, les étiquettes temporaires sont mémorisée dans une structure `Q`. Cette structure doit à la fois permettre d'accéder au plus petit élément (donc idéalement une file de priorité), mais aussi à toutes les étiquettes associées à un nœud (donc idéalement un tableau de listes d'étiquettes).

À cause du besoin d'accéder de manière efficace de ces manières, il n'est pas possible d'utiliser des conteneurs classique de STL. Pour cela nous avons utilisé Boost Multiindex qui permet de gérer plusieurs indexes permettant d'avoir accès efficacement aux étiquettes des deux manières souhaitées.

A.2.2 Gestion des horaires

Si au premier abord gérer les grilles horaires est trivial, l'application est un peu plus délicate. Il est en effet nécessaire de prendre en compte que les bus et les trains ne circulent pas tous les jours aux mêmes horaires. Pour les transports en commun urbains il existe en France généralement cinq cas différents : jour de semaine, samedi, dimanche, vacances, premier mai. Cependant ne considérer que ces cinq cas est restrictif. Ainsi la SNCF mettra également en place des trains supplémentaires les jours de départ de vacances. De ce fait il est nécessaire de gérer au cas par cas.

Comme tout au long de cette thèse, nous avons favorisé l'approche la plus générique. De ce fait nous considérons que chaque jour le plan de circulation peut être différent. Cependant pour des raisons de performance, nous ne pouvons pas nous permettre d'effectuer des comparaisons de dates au parcours de chaque arc.

Nous avons donc fait le choix que l'horizon de dates est fini. Pour chaque arc nous avons un tableau de triplets (instant de départ, instant d'arrivée, dates valides). Les dates valides sont représentés par un bitset (qui peut être considéré comme un tableau de booléens). Ainsi avec un horizon de sept jours commençant le premier janvier, le bitset `0011001` indiquera que l'horaire est valable les 3, 4 et 7 janvier. La place pour avoir un horizon de deux mois est de huit octets. Cela reste donc acceptable en terme de consommation de mémoire.

Les dates sont triées par ordre croissant afin de pouvoir arrêter la recherche dès le prochain départ pour la date souhaitée est trouvée. Il n'est malheureusement pas possible de faire une recherche par dichotomie puisqu'il n'est pas possible de faire une bissection pour un jour donné. Par exemple si le dimanche les bus s'arrêtent à 19h tandis qu'en semaine ils circulent jusqu'à minuit, une bissection à 20h sera problématique.

A.3 Capture d'écran du démonstrateur

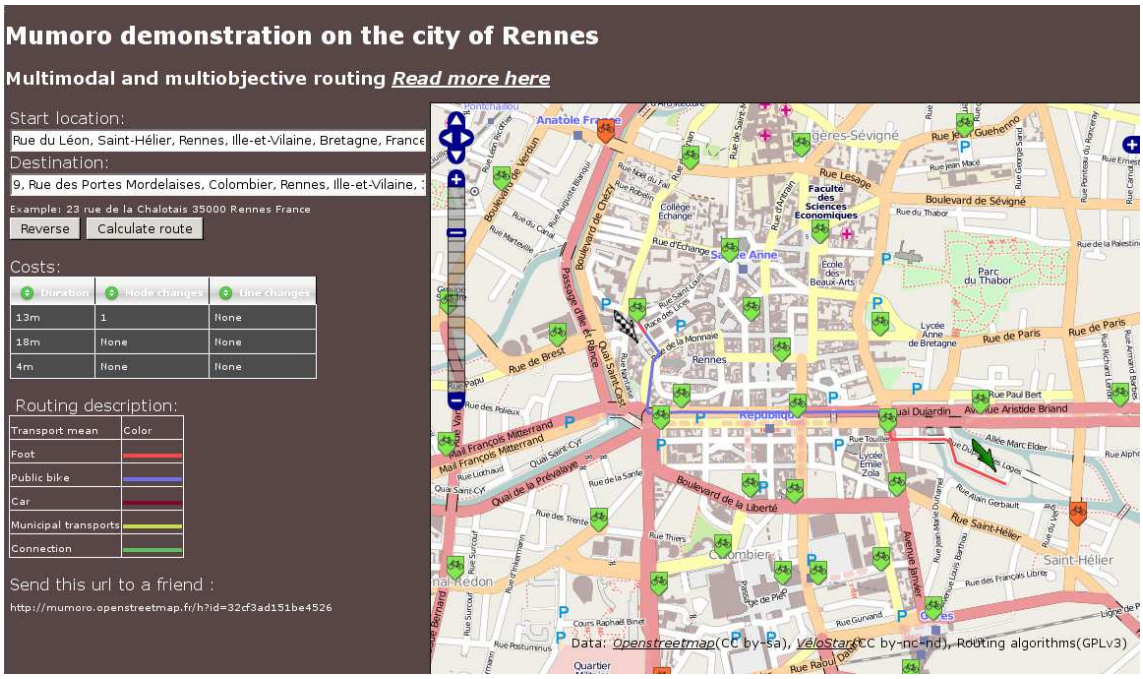


Figure A.1 Capture d'écran du démonstrateur développé

Annexe B
Systèmes existants

Les calculateurs d'itinéraires sont largement répandus. Cependant, ils ne présentent souvent qu'un intérêt limité. Nous présentons donc quelques réalisations remarquables.

B.1 Navitia de Canal TP

Canal TP est une entreprise française spécialisée dans les systèmes d'information pour le transport public. Leur application phare est *Navitia* qui est un calculateur d'itinéraires web très largement utilisé en France. Il couvre notamment tous les transports en commun de l'Île-de-France.

Le calculateur prend en compte la marche et propose d'optimiser pour avoir le trajet le plus rapide, avec le moins de marche ou encore le moins de correspondances. Il ne permet cependant pas d'optimisation multiobjectif et la marche n'est considérée que pour rejoindre un arrêt (ainsi le système ne proposera pas de marcher pendant trente minutes s'il n'existe pas de meilleure connexion).

B.2 Reittiopas

Le site Reittiopas[21] est le calculateur d'itinéraire de l'agglomération de Helsinki (Finlande). Il propose tous les transports en commun, la marche et le vélo. Il propose d'optimiser soit le temps, les changements ou encore la distance de marche sans être multiobjectif.

Ce système a l'avantage de considérer la marche comme un mode à part entière. Ainsi, dans certains cas (par exemple la nuit), l'itinéraire proposé sera effectué majoritairement à pied.

[21] `http://www.reittiopas.fi/en/`

B.3 Google maps

Dans son outil de cartographie web[22], Google propose différents calculs d'itinéraires de point à point en obligeant à choisir le mode exclusif (voiture, marche et dans certaines villes le vélo et les transport en commun).

Avec les transports en commun, aucun objectif ne peut être défini, mais plusieurs alternatives sont proposées. Dans certains cas, la marche est proposée, mais pas systématiquement.

B.4 Géovélo

Géovélo[23] est un calculateur d'itinéraire à vélo développé par Gaël Sauvanet dans le cadre de sa thèse Sauvanet,Néron [58]. Il s'agit d'un calculateur multiobjectif et propose plusieurs trajets qui sont un compromis entre la distance et le sentiment de sécurité.

B.5 Transpolitan

Transpolitan est une entreprise spécialisée dans le transport multimodal et propose un calculateur d'itinéraire. Les captures d'écran indiquent qu'il devait s'agir d'un calculateur multiobjectif (`http://www.transpolitan.com/pvm.htm`). Cependant il semble que le site soit abandonné depuis plusieurs années et il est difficile d'avoir des informations sur une éventuelle application réelle.

B.6 OpenTripPlanner

Le projet OpenTripPlanner[24] vise à fédérer plusieurs projets libres dans le domaine du calcul d'itinéraire afin de développer une boîte à outil générique.

En ce qui concerne le calcul d'itinéraire à proprement parler, ils utilisent le programme *Graphserver*[25] qui est un calculateur multimodal sous licence libre.

Son avantage est la généricité (il n'y a aucune supposition sur la nature de l'itinéraire souhaité par l'utilisateur). Cependant il ne permet que de calculer l'itinéraire le plus rapide. L'approche *time-expanded* utilisée pour les transports en commun et le code fait en C rendent une extension des algorithmes délicate.

---

[22] `http://maps.google.com`

[23] `http://www.geovelo.fr/`

[24] `http://opentripplanner.org`

[25] `http://bmander.github.com/graphserver/`

Bibliographie

1 Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269–271.

2 Fredman, M. and Tarjan, R. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM), 34*(3), 596–615.

3 Hart, P., Nilsson, N. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics, 4*(2), 100–107.

4 Schultes, D. (2007). Route planning in road networks. *Science, 316*(5824), 566.

5 Geisberger, R., Sanders, P., Schultes, D. and Delling, D. (2008). Contraction hierarchies : Faster and simpler hierarchical routing in road networks. *Experimental Algorithms*, pp. 319–333.

6 Sanders, P. and Schultes, D. (2005). Highway hierarchies hasten exact shortest path queries. *Lecture Notes in Computer Science, 3669*, 568–579.

7 Goldberg, A. and Harrelson, C. (2005). Computing the shortest path : A search meets graph theory. , pp. 156–165.

8 Bast, H., Funke, S., Matijevic, D., Sanders, P. and Schultes, D. (2007). In transit to constant time shortest-path queries in road networks. , pp. 46–59.

9 Hilger, M., Köhler, E., Möhring, R. and Schilling, H. (2009). Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem : Ninth DIMACS Implementation Challenge*, pp. 73–92.

10 Bauer, R., Delling, D., Sanders, P., Schieferdecker, D. and Schultes, D. et al. (2008). Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. *Lecture Notes in Computer Science, 5038*, 303.

11 Delling, D., Sanders, P., Schultes, D. and Wagner, D. (2009). Engineering route planning algorithms. *Algorithmics of Large and Complex Networks, Lecture Notes in Computer Science. Springer*.

12 Abraham, I., Fiat, A., Goldberg, A. and Werneck, R. (2010). Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. , *22*, 47–55.

13 Dreyfus, S. (1969). An appraisal of some shortest-path algorithms. *Operations Research, 17*(3), 395–412.

14 Gondran, M. and Minoux, M. (2002). Graphes, Dioïdes et semi-anneaux. *TEC & DOC, Paris*.

15 Nannicini, G., Delling, D., Liberti, L. and Schultes, D. (2008). Bidirectional A* Search for Time-Dependent Fast Paths. *Lecture Notes in Computer Science, 5038*, 334.

16 Delling, D. (2008). Time-dependent SHARC-routing. *Algorithmica*, pp. 1–35.

17 Batz, V., Delling, D., Sanders, P. and Vetter, C. (2009). Time-dependent contraction hierarchies. , pp. 97–105.

18 Hizem, M. (2008). *Recherche de chemins dans un graphe à pondération dynamique*. PhD thesis, École Centrale de Lille.

19 Muller-Hannemann, M., Schulz, F., Wagner, D. and Zaroliagis, C. (2007). Timetable information : Models and algorithms. *Lecture Notes in Computer Science, 4359*, 67.

20 Pallottino, S. and Scutella, M. (1998). *Shortest path algorithms in transportation models : classical and innovative aspects*. Kluwer Academic Publishers.

21 Bast, H., Carlsson, E., Eigenwillig, A., Geisberger, R. and Harrelson, C. et al..Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns. *Algorithms–ESA 2010*, pp. 290–301.

22 Geisberger, R. (2009). Contraction of Timetable Networks with Realistic Transfers. *Arxiv preprint arXiv :0908.1528*.

23 Pyrga, E., Schulz, F., Wagner, D. and Zaroliagis, C. (2004). Experimental comparison of shortest path approaches for timetable information. , pp. 88–99.

24 Delling, D., Pajor, T., Wagner, D. and Karlsruhe, G. (2009). Engineering Time-Expanded Graphs for Faster Timetable Information. *Robust and Online Large-Scale Optimization : Models and Techniques for Transportation Systems*, pp. 182.

25 Orda, A. and Rom, R. (1991). Minimum weight paths in time-dependent networks. *Networks, 21*(3), 295–319.

26 Ahuja, R., Orlin, J., Pallottino, S. and Scutella, M. (2002). Minimum Time and Minimum Cost-Path Problems in Street Networks with Periodic Traffic Lights. *Transportation Science, 36*(3), 326–336.

27 Chabini, I. (1998). Discrete dynamic shortest path problems in transportation applications : Complexity and algorithms with optimal run time. *Transportation Research Record : Journal of the Transportation Research Board, 1645*(-1), 170–175.

28 Chabini, I. and Ganugapati, S. (2002). Parallel algorithms for dynamic shortest path problems. *International Transactions in Operational Research, 9*(3), 279–302.

29 Coello, C., Van Veldhuizen, D. and Lamont, G. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer.

30 Ehrgott, M. (2000). *Multicriteria optimization*. Springer-Verlag.

31 Ehrgott, M. and Gandibleux, X. (2000). A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectrum, 22*(4), 425–460.

32 Raith, A. and Ehrgott, M. (2009). A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research, 36*(4), 1299–1331.

33 Clímaco, J. and Pascoal, M. (2010). Multicriteria Path and Tree Problems–Discussion on Exact Algorithms. .

34 Muller-Hannemann, M. and Weihe, K. (2001). Pareto Shortest Paths is Often Feasible in Practice. *Lecture Notes In Computer Science*, pp. 185–198.

35 Galand, L. and Perny, P. (2006). Search for compromise solutions in multiobjective state space graphs. , pp. 93.

36 Grabish, M. (2006). L'utilisation de l'intégrale de Choquet en aide multicritère à la décision. *European Working Group "Multiple Criteria Decision Aiding, 3*, 5–10.

37 Fouchal, H., Gandibleux, X. and Lehuédé, F. (2010). Algorithme de Martins et intégrale de Choquet pour le calcul de plus courts chemins multi-critères préférés. *ROADEF 2010*.

38 Galand, L., Perny, P. and Spanjaard, O. (2010). Choquet-based optimisation in multiobjective shortest path and spanning tree problems. *European Journal of Operational Research, 204*(2), 303–315.

39 Gabrel, V. and Vanderpooten, D. (2002). Enumeration and interactive selection of efficient paths in a multiple criteria graph for scheduling an earth observing satellite. *European Journal of Operational Research, 139*(3), 533–542.

40 Galand, L. (2006). Interactive search for compromise solutions in multicriteria graph problems. , pp. 22–25.

41 Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm : NSGA-II. *IEEE transactions on evolutionary computation, 6*(2), 182–197.

42 Zitzler, E., Laumanns, M., Thiele, L. and others (2001). SPEA2 : Improving the strength Pareto evolutionary algorithm. , pp. 95–100.

43 Coello, C., de Computacion, S. and Zacatenco, C. (2006). 20 Years of Evolutionary Multi-Objective Optimization : What Has Been Done and What Remains To Be Done. .

44 Maria, J., Pangiliana, A. and Janssens, G. (2007). Evolutionary algorithms for the multiobjective shortest path planning problem. *International journal of computer and information science and engineering*, *1*(1), 54–59.

45 Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *Neural Networks Proceedings., IEEE International Conference on*, *4*.

46 Durillo, J. J., García-Nieto, J., Nebro, A. J., Coello, C. A. C. and Luna, F. et al. (2009). Multi-Objective Parcticle Swarm Optimizers : An Experimental Comparison. , pp. 495–509.

47 Dorigo, M., Maniezzo, V. and Colorni, A. (1996). The ant system : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, *26*, 29–41.

48 Alaya, I., Solnon, C. and Ghedira, K. (2007). Ant Colony Optimization for Multi-objective Optimization Problems. , pp. 450–457.

49 Dorigo, M. and Stützle, T. (2004). *Ant colony optimization*. MIT press.

50 Pluciński, M. (2005). Application of the ant colony algorithm for the path planning. .

51 Kolavali, S. and Bhatnagar, S. (2009). Ant Colony Optimization Algorithms for Shortest Path Problems ?. , pp. 37.

52 Hansen, P. (1980). Bicriterion path problems. , pp. 109.

53 Martins, E. (1984). On a multicriteria shortest path problem. *European Journal of Operational Research*, *16*(2), 236–245.

54 Guerriero, F. and Musmanno, R. (2001). Label correcting methods to solve multicriteria shortest path problems. *Journal of Optimization Theory and Applications*, *111*(3), 589–613.

55 Tarapata, Z. (2007). Selected multicriteria shortest path problems : An analysis of complexity, models and adaptation of standard algorithms. *International Journal of Applied Mathematics and Computer Science*, *17*(2), 269–287.

56 Stewart, B. and White III, C. (1991). Multiobjective A*. *Journal of the ACM (JACM)*, *38*(4), 814.

57 Mandow, L. and de la Cruz, J. (2005). A New Approach to Multiobjective A* Search. , *19*, 218.

58 Sauvanet, G. and Néron, E. (2009). Calcul de plus court chemin bicritére. *ROADEF 2009*, pp. 90.

59 Delling, D. and Wagner, D. (2009). Pareto Paths with SHARC. , *5526*, 125–136.

60 Geisberger, R., Kobitzsch, M. and Sanders, P..Route Planning with Flexible Objective Functions. .

61 Berger, A., Grimmer, M. and Müller-Hannemann, M. (2010). Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks. *Experimental Algorithms*, pp. 35–46.

62 Muller-Hannemann, M. and Schnee, M. (2007). Finding all attractive train connections by multi-criteria Pareto search. *Lecture Notes In Computer Science, 4359*, 246.

63 Papadimitriou, C. and Yannakakis, M. (2000). On the Approximability of Trade-offs and Optimal Access of Web Sources (Extended Abstract). , pp. 86.

64 Warburton, A. (1987). Approximation of Pareto optima in multiple-objective, shortest-path problems. *Operations Research*, pp. 70–79.

65 Tsaggouris, G. and Zaroliagis, C. (2005). Improved FPTAS for multiobjective shortest paths with applications. *CTI Techn. Report TR-2005/07/03*.

66 Boussedjra, M., Bloch, C. and El Moudni, A. (2003). Solution optimale pour la recherche du meilleur chemin intermodal. *MOSIM*.

67 Ziliaskopoulos, A. and Wardell, W. (2000). An intermodal optimum path algorithm for multimodal networks with dynamic arc travel times and switching delays. *European Journal of Operational Research, 125*(3), 486–502.

68 Bast, H. (2009). Car or Public Transport—Two Worlds. *Efficient Algorithms*, pp. 355–367.

69 Lozano, A. and Storchi, G. (2001). Shortest viable path algorithm in multimodal networks. *Transportation Research Part A, 35*(3), 225–241.

70 Delling, D., Pajor, T. and Wagner, D. (2009). Accelerating Multi-Modal Route Planning by Access-Nodes. .

71 Barrett, C., Bisset, K., Holzer, M., Konjevod, G. and Marathe, M. et al. (2008). Engineering Label-Constrained Shortest-Path Algorithms. *Lecture Notes in Computer Science, 5034*, 27–37.

72 Meng, F., Yizhi, L., Wai, L. and Chuin, L. (1999). A Multi-Criteria, Multi-Modal Passenger Route Advisory System. .

73 Boussedjra, M., Bloch, C. and El Moudin, A. (2006). Apport d'une technique de diversification dans un algorithme de recherche de chemin intermodal. *MOSIM*.

74 Gueye, F., Artigues, C. and Huguet, M. (2009). Planification d'itinéraires en transport multimodal. *ROADEF 2009*, pp. 113.

75 Gueye, F., Artigues, C., Huguet, M., Schettini, F. and Dezou, L. (2010). A bidirectional/multi-queue algorithm for the bi-objective multimodal viable shortest path problem. .

76 Cordeau, J., Laporte, G., Potvin, J. and Savelsbergh, M. (2007). Transportation on demand. *Transportation, 14*, 429–466.

77 Baldacci, R., Maniezzo, V. and Mingozzi, A. (2004). An exact method for the car pooling problem based on Lagrangean column generation. *Operations Research*, pp. 422–439.

78 MacQueen, J. and others (1966). Some methods for classification and analysis of multivariate observations. Western Management Science Inst Unvi Of California Los Angeles.